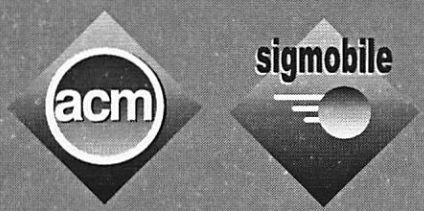
conference

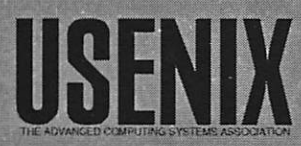
proceedings

Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services

Seattle, WA, USA
June 5, 2005

Jointly sponsored by
ACM SIGMOBILE and
The USENIX Association,
in cooperation with **ACM SIGOPS**





For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
URL: http://www.usenix.org

The price is $10.
Outside the U.S.A. and Canada, please add
$5 per copy for postage (via air printed matter).

ISBN 1-931971-32-3

USENIX Association

Proceedings of the Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR '05)

Jointly sponsored by USENIX and ACM SIGMOBILE,
in cooperation with ACM SIGOPS

June 5, 2005
Seattle, WA, USA

Conference Organizers

EESR '05 Workshop Co-Chairs

Ron Ambrosio, *IBM Research, USA*
Chatschik Bisdikian, *IBM Research, USA*

EESR '05 Workshop Program Committee

Phillippe Bonnet, *University of Copenhagen, Denmark*
Deborah Estrin, *University of California, Los Angeles, USA*
Michael J. Franklin, *University of California, Berkeley, USA*
Rick Han, *University of Colorado, Boulder, USA*
Wei Hong, *Intel Research, USA*
Holger Karl, *University of Paderborn, Germany*
Dritan Kaleshi, *University of Bristol, UK*
Amy L. Murphy, *University of Lugano, Switzerland*
Priya Narasimhan, *Carnegie Mellon University, USA*
Joe Paradiso, *MIT Media Lab, USA*
Yannis Paschalidis, *Boston University, USA*
Salil Pradhan, *HP Labs, USA*
Johnathan M. Reason, *IBM Research, USA*
Kay Römer, *ETH Zurich, Switzerland*
Gaurav Sukhatme, *University of Southern California, USA*
Nalini Venkatasubramanian, *University of California, Irvine, USA*
Feng Zhao, *Microsoft Research, USA*

The USENIX Association Staff

Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR '05)

June 5, 2005

Seattle, WA, USA

Sunday, June 5, 2005

Sense 'n Respond Technologies

Sense 'n Respond Solutions

Index of Authors

Message from the Workshop Co-Chairs

It has been long realized that the development and exploitation of intelligent sensors and actuators and their networks (S&As and SANETs, respectively) will enable existing operations and processes to become more efficient, nimble, and capable to react to new situations dynamically, and, at the same time, spearhead a new generation of applications and services not previously imagined. So it comes as no surprise that a large and diverse set of organizations from retail store heavyweights to governmental (at the local and national level) agencies, from emergency responders to utility operators, agriculturalists, habitat scientists, and so on, are all looking into ways of incorporating SANETs in their processes to improve their operations, reduce cost, provide higher customer service, and increase yield.

Any search, casual or intentional, of scholarly journals and technical conferences reveals a very large number of forums where SANET-related technologies are being presented and debated. A large majority of these forums exhibit an introvert interest in the SANETs themselves, focusing in their internal operation covering areas like: hardware and software technologies for S&As, S&A design architectures, energy considerations, physical, medium access, and networking protocols, ad hoc operation, SANET-facing middleware (that manage connectivity, mobility, power, auto-configuration, etc.), information flow within a SANET, and so on.

While the above research activities are very important because they establish the foundations through which SANETs will deliver the intelligent capabilities anticipated of them, developing efficient and intelligent SANET technologies is not the end-game. The endgame is the integration of those SANET technologies into end-to-end solutions that provide some measure of value (whether business or scientific).

We foresee that SANET-enabled application and services will have to cope with many more issues than just establishing themselves and managing their internal flow of information. As an example, it has been said that one of the major hindrances in the widespread adoption of RFID technologies for asset tracking is not the technologies themselves, but the overwhelming cost to an enterprise of "aligning" their legacy data applications to cope with a new data format! Not surprising, the ages-old problem of data heterogeneity emerges once again in the SANET application space. This problem is further exacerbated when applications must deal with multiple SANETs, introducing heterogeneity issues, resource management across multiple SANETs, quality of data issues, SANET services brokering, and so on.

In this workshop we brought together researchers, professionals, and practitioners working in the areas related to the aforementioned aspects of developing end-to-end SANET-enabled applications and services. What we believe set this workshop apart from other venues for related topics is that it focused on SANETs as components of end-to-end SANET-enabled applications and services. It brought to the discussion table new aspects such as: how SANETs (and more generally the large-scale sense and respond control systems they enable) can become effective parts of business processes (such as RFID-enabled supply chain systems, building energy optimization processes, manufacturing systems, traffic management systems, power utility operations, etc.) to support intelligent applications and services.

The workshop served as a forum to expose, formalize, discuss and address (some of) these aspects. It also served as a forum to entice additional researchers to explore the challenging topics that the SANET area faces on its way to becoming an integral and pervasive part of the way we live and work.

We received many submissions covering a number of issues pertinent to the scope of the workshop including: coping with heterogeneity of S&A technologies; architectures and technologies for next-generation/emerging SANET-enabled application and services; programming models for SANET systems; application-facing middleware for SANET-enabled applications and services; development and deployment of S&A-enhanced applications; integration of sensor technologies in business processes; and so on. While we could not accommodate all these submissions in our workshop, the papers you find in these proceedings represent excellent examples of key research challenges and their potential solutions for building end-to-end S&A-aided solutions.

We would like to thank the many authors who submitted papers to our workshop and congratulate the ones whose papers are included in these proceedings. We hope they all continue their work in this challenging and exciting research area, and that we see their work appear in one of the many future technical meetings we anticipate will be held in the areas pertinent to our workshop.

In closing, the Co-Chairs would like to thank the generous efforts of our Program Committee, the workshop participants, and the support staff of USENIX in making this a successful and beneficial endeavor.

Ron Ambrosio & Chatschik Bisdikian, IBM T.J. Watson Research Center
Workshop Co-Chairs

A File System Abstraction for Sense and Respond Systems

Sameer Tilak[1], Bhanu Pisupati[2], Kenneth Chiu[1], Geoffrey Brown[2], Nael Abu-Ghazaleh[1]
[1]*Computer Science Department, State University of New York (SUNY) at Binghamton*
[2]*Computer Science Department, Indiana University*

***Abstract*— The heterogeneity and resource constraints of sense-and-respond systems significantly challenge system and application development. In this paper, we present a flexible, intuitive file system abstraction for organizing and managing sense-and-respond systems based on the Plan 9 design principles. A key feature is the support of multiple views of the system via filesystem namespaces. Constructed logical views present an application-specific representation of the network, thus enabling high-level programming. Concurrently, structural views of the network enable resource-efficient planning and execution of tasks. We present and motivate the design using several examples, outline research challenges and our plan to address them, and describe the current implementation state.**

I. Introduction

The heterogeneity and resource constraints of typical sense-and-respond (S&R) systems pose daunting challenges to system and application development. These challenges are further exacerbated by the lack of simple abstractions for the use and development of these systems. In this paper, we show how the principles of Plan 9 [1] can be applied to S&R systems, resulting in flexible, intuitive systems supporting multiple logical views. Applications can then use the view with the most appropriate organization and abstraction.

Sense-and-respond systems typically comprise a diverse set of hardware and software elements. Hardware elements include a wide variety of different sensor and actuator types, ranging from COTS to highly-specialized, one-of-a-kind parts. Software elements draw from numerous domains, including the natural sciences, artificial intelligence, sensor networks, and embedded systems. Further increasing the diversity is the various ways in which the software and hardware elements may interact, such as event-driven, polled data, or data streams. This heterogeneity greatly complicates the development of reliable, effective S&R systems.

A crucial component of many S&R systems is wireless sensor and actuator networks. These networks promise to revolutionize sensing in a wide range of civil, scientific, military, and industrial applications. For example, numerous sensors may be deployed around a city to monitor chemical and biological threats, or in the wild to monitor ecological events in migration patterns [2],

This work was partially supported by NSF Award SCI-0330568, NSF Award DBI-0446298, and NSF Award EIA-0202048.

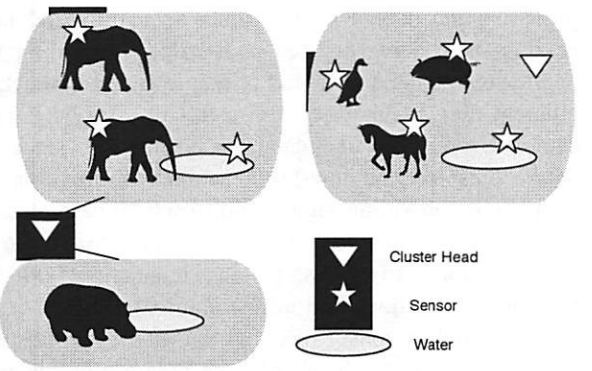


Fig. 1. An example wireless sensor network in a zoo. Sensors track animal locations and resources such as food and water. The network is divided into two clusters, each consisting of a cluster head.

or to track a smoldering forest fire for conditions that might lead to an outbreak. Responses may range from alerts to the use of actuators to mitigate the damage.

The inherent resource constraints of WSNs pose significant challenges to this vision. Wireless sensors are limited in power, weight, and size; also, communication is often unreliable. These constraints exacerbate the problems created by the heterogeneity of S&R systems.

Successfully addressing these multi-dimensional challenges relies crucially on developing an effective abstraction for sensor networks. A simple and well understood abstraction can significantly ease both system and application development. Many sensor networks are deployed by scientists and researchers whose expertise is not computer science. Providing these scientists with a simple and intuitive interface to program, access, configure, and debug sensors can facilitate deployment of large scale sensor networks to a great extent.

Motivated by this need, we propose a simple yet powerful filesystem-based abstraction of sensor networks based on Plan 9, which espouses that the file system metaphor (as seen, for example, in the `/proc` file system) can be adopted for almost all aspects of system design and development. Not only can files be used to store a named sequence of bytes, but also to replace many aspects of communication and control that are typically performed using system calls. A key feature of our proposed solution is the ability of the application to define multiple namespaces to organize the sensor

network in an application specific manner. Another advantage is that we can now exploit, perhaps with some adaptation, much of the work in distributed file systems, such as Coda [3] to address various systems issues such as consistency models.

Another commonly proposed abstraction of WSNs is that of a database [4]. Typically, these databases present application-level information, isolated from the resources providing the data. Upstream data acquistion and processing then lacks resource knowledge, precluding the application of the end-to-end principle and complicating efficient implementations. Infrastructure services also cannot be built on the database abstraction, since by nature these require low-level resource information. In contrast, by providing logical and structural namespaces, our file system abstraction can present information at a wide variety of levels, making it suitable both for building applications and infrastructure.

We model a sensor network as a set of clusters, each with a cluster head. Cluster membership is normally determined geographically. Our model is intended merely to provide a concrete basis for demonstrating the utility of our file system abstraction, and not as an end in itself. With this abstraction, an application might access sensor data geographically by reading from a path `/location/54W/35N/data`, or logically such as `/data/temperature/snakes`.

This paper contributes a file system abstraction for sensor networks and a proof-of-concept implementation within the ns-2 simulator. The Plan 9 protocol for implementing the file system abstraction, Styx, has already been well-researched on various distributed computing platforms. We thus focus our attention on its implementation in sensor networks.

II. Proposed Filesystem Abstraction

The application of file system abstractions to sensor networks is inspired by the tenets of the Plan 9 and Inferno operating systems [1][5], whose defining feature is the uniform treatment of devices and files. This section describes the use of the file system abstraction as a convenient and efficient means to view, access and program sensor networks.

Figure 1 shows a sample network deployed in a zoo, with sensors tracking animals and resources such as food and water. The network is divided into two clusters each consisting of a cluster head. The sensors themselves may each differ in functionality (temperature/position), hardware type (MSP/AVR), and software platform. Figure 2 shows a typical directory layout for such a network.

The file system representation naturally captures the structure of the network in addition to depicting logical attributes such as aggregation properties and groupings. The root directory `network` encapsulates the whole

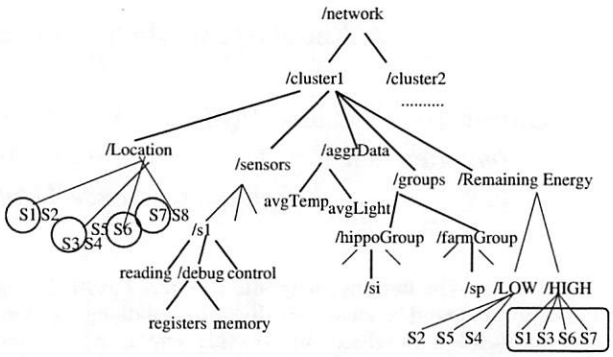


Fig. 2. Namespace for a sensor network.

network. It has a subdirectory for each cluster, each which in turn has three subdirectories named `sensors`, `aggrData`, and `groups`. The `sensors` directory provides direct access to the sensors and has one directory corresponding to each. A lot of sensor network applications are data centric and often however, rather than the individual sensor values, what is of interest is the aggregate value of a property observed at different sensors. These cluster-wide properties can be readily retrieved via the `aggrData` directory, which contains files `avgTemp` and `avgLight`, representing the average temperator and average light readings, respectively. We thus embed "intelligence" into the file system, enabling it to process and interpret data (such as averaging individual readings), rather than just storing and presenting it. Finally, the `groups` directories demonstrate the logical grouping of the sensors according to specific criteria. The grouping shown is based on animal type, but could have been based on geographic location of the sensors.

The task of locating and denoting a sensor device effectively reduces to finding the path for its corresponding file in the namespace. Sensor 1's value, for instance, is read from `/network/cluster1/sensors/s1/reading`. The low level operations inherent in retrieving the values are cleanly abstracted by the file interface, which also conceals heterogeneity among sensors.

Another example of using this interface is that of configuration and debugging of sensors. The file system can translate writes to the `control` file to control operations on the sensor, such as reset, wakeup, or sleep. The file system may also facilitate debugging by exposing the sensors' `registers` and `memory` as files. An external debugger can then use the file system interface to debug software executing on the sensors. described in the next section.

The file system approach can flexibly partition functionality at different levels of the network, such as at the sensor, cluster head, or client; thus providing a single paradigm even for end sensor devices that may be

computationally lightweight. Logically combining multiple networks now becomes analogous to mounting the networks' file system representations under a common directory.

III. Architecture

As in Plan 9 and Inferno [1][5], all resources are named and accessed as files within a hierarchical directory structure, implemented using the Styx protocol from Inferno[6]. Systems can overlay arbitrarily complex data and sensor policies using multiple simultaneous namespaces, each providing a different perspective of the same physical sensor network.

A client that wishes to interact with a sensor network mounts the associated file system from a server and executes the appropriate file operations. The client and server interact using the Styx messaging protocol to encode the various file operations. Message are always exchanged in pairs, with the client initiating the exchange and the server responding. The client starts a session by connecting to a server using a *Tattach* message. The client may then navigate the directory tree using the *Twalk* message (analogous to the UNIX `cd` command). Other standard operations such as opening, reading, and writing to files are performed using the *Topen*, *Tread*, and *Twrite* messages respectively. These operations may block, but multiple outstanding requests may be issued to compensate.

Client applications do not directly issue Styx messages, but rather use a software library we provide. The interface consists of typical file operations such as *open*, *read*, and *write*. Details of the underlying messaging protocol are completely concealed from the client application.

The file server implementation of sensor networks has two components in its core, namely device-level file servers and multiplexers. Device-level servers reside in the leaf sensors, and define a static directory structure and methods for accessing individual files (really named resources). These fundamental servers provide the most basic interactions with the sensors such as reading the value and primitive control operations. Correspondingly, these servers store minimal dynamic state about themselves and and active clients, and hence require limited runtime memory.

Multiplexers merge different device-level file systems, and would typically reside in the cluster heads to provide a cluster-level namespace. At startup time, the multiplexer engages a discovery process to determine the topology of its associated sensors. It then reads the static directory structure from the device level file systems of all sensors to create cluster-level file system hierarchy. When a client requests a file operation, the multiplexer uses the file descriptor in the request to map (multiplex) and reissue the request to a particular device file system in its namespace. Multiplexers can receive and process new requests while waiting for a reply from an outstanding request to a device file server. A multiplexer also has other responsibilities including collection of data from multiple sensors and applying aggregation functions (such as average) to it. It manages logical groups of sensors shown in the `groups` directory in Figure 2. Sensors are sorted into groups during startup and also afterwards when new sensors come online.

Multiplexers offer great flexibility in partitioning application, configuration, and debugging functionality between different components of the sensor network. Consider a debugger application that is debugging code executing on a sensor node. The debugger typically requires access to registers and memory on the sensor. Instead of implementing the low-level functionality to retrieve these values in the debugger itself, the functionality can instead be implemented in the device-level servers as files, with the multiplexer in the cluster head providing higher-level organization. The debugger then accesses the registers and memory indirectly by reading and writing to these files, upon which the device-level server performs the necessary low level procedures. As another example, migration from a simulated sensor network to a real network is straightforward. The device file servers can present the same file interface to the application regardless of whether the server is accessing a simulated sensor or a real sensor.

IV. Research Challenges

We have identified following research challenges specific to using the file system abstraction in a sensor network environment.

Supporting resource efficient operation: Abstraction such as those mentioned in Section II hide complexities of the underlying system, and provide rich and intuitive interfaces to the end user. Since wireless sensors are often resource-constrained, however, the implementation of the file system abstraction must be reasonably efficient; and thus the protocols must be designed to operate within severe constraints on computational power, energy, and storage. We selected the Styx protocol in part because it is lightweight, and does not impose excessive overhead, as detailed in Section VII.

Application-level operations must also be resource-efficient, and so we propose the construction of a standard resource namespace that exposes resource information to the application, such as the available energy or storage space on a given sensor node. The following challenges are identified.

Supporting resource efficient operation: Abstraction such as those mentioned in Section II hide complexities of the underlying system, and provide rich and intuitive

interfaces to the end user. Since wireless sensors are often resource-constrained, however, the implementation of the file system abstraction must be reasonably efficient; and thus the protocols must be designed to operate within severe constraints on computational power, energy, and storage. We selected the Styx protocol in part because it is lightweight, and does not impose excessive overhead, as detailed in Section VII.

Application-level operations must also be resource-efficient, and so we propose the construction of a standard resource namespace that exposes resource information to the application, such as the available energy or storage space on a given sensor node. Section VI describes an example of resource-efficient query execution.

While the filesystem abstraction provides mechanisms for creating and maintaining namespaces, it does not define how they should be organized (separation of policy from mechanism).

Consistency models: By nature, WSNs are dynamic, concurrent systems. Thus, clients' view of the namespace and even data may be inconsistent with respect to the current actual state. For example, a client may use the namespace to determine that a particular mobile sensor is in a specific region, but discover when it actually reads data from the sensor, that it has moved out of the previously determined region. Or, stale, cached sensor readings may be sent to a client as a result of transmission interruptions.

Strong consistency models could be implemented using distributed locks and other techniques, but the nature of WSN applications generally suits weak consistency models. Sensor data is by nature unreliable, and applications usually do not rely on high-quality, consistent operation. Adopting the file system abstraction, also allows us to apply existing research in distributed file system consistency models [3] to sensor network domain.

Managing streaming data: Sensors typically produce stream data representing their samples over time. Stream data is not directly supported in the Styx framework; extensions to the file system abstraction to model streaming devices may be required. We are currently working on extending Styx protocol so that streaming data can be handled more efficiently.

Supporting in-network application-specific processing: Our framework supports in-network aggregation in the following two ways. In the first approach, a user can extend the existing Styx server to incorporate the required functionality. The Styx server implementation is fairly simple and easy to extend. In the second approach, the user implements the functionality within an independent dynamic library. The Styx server then loads the dynamic library at run time as needed, and unloads them when unneeded to free up memory.

For applications with relatively static requirements, such as a debugging application which needs access to sensor registers and memory, the first design choice is a better option. Also, very commonly used aggregation functions including average, min, max can be implemented within the Styx server. However, for applications that require more sophisticated in-network processing or whose functionality changes more often, the second design choice is a better option.

Tolerating network unreliability: Wireless channels are susceptible to fading and interference. Furthermore, to conserve energy, sensors often turn off their radios for extended periods of time. This intermittent connectivity poses unique challenges to filesystem design. One partial solution is to cache relatively static sensor information in the cluster head, which can then respond to queries even when communication to the sensor is interrupted.

V. Additional Capabilities

Using a file system abstraction offers additional advantages for application developers in a sensor network. Some of these are reviewed in this section.

Ease of application development: The file system interface is well understood (both semantically and syntactically) by application developers and system programmers. This interface can be easily used by scientists and researchers who are not familiar with the intricacies and low-level details of sensor network systems.

Access control via file permissions: File systems incorporate simple but flexible access control mechanisms via file permissions. For example, the permissions on a sensor control file might grant write access to the administrator group to allow calibration, while only granting read access to normal users to allow querying the current device state.

Ease of integration: We believe that tools designed in other contexts can be easily made available to use in sensor network environments. This includes, for example, development and visualization tools developed for desktops, PDAs, or even distributed systems. These tools can then be ported to the proposed file system abstraction with an effort significantly lower than having to develop them from scratch.

Portability across sensor architectures and protocols: The file system abstraction using the Styx protocol can serve as a bridging layer for interoperating heterogeneous sensors as well as interactions with external devices. In this sense, it plays a role similar to that played by IP in interconnecting heterogeneous networks. Once a new device has support for file system/Styx primitives, it is able to interoperate with the remainder of the system.

VI. Examples

In this section we demonstrate the capabilities of the file system abstraction with three examples.

A. Sensor Monitoring and Calibration

Monitoring the resource state of sensors is an important capability for sensor networks [7]. Moreover, sensor calibration is essential for reducing the noise in the sensor data [8]. The file system provides mechanisms to discover sensors, as well as read and write their state, which allow the application developers to rapidly and even interactively monitor and calibrate the sensor network. For example, the following commands can be issued by a client to discover the temperature sensors in an area, read the remaining energy of one of the sensors, and then write a parameter to calibrate another.

```
mount /dev/network /network
ls /network/cluster1/sensors/
cat /network/cluster1/s1/remaining-energy
echo 2.5 > /network/cluster1/s1/control
```

Note that an application-specific namespace can provide S&R functionality similarly to the example above. We may monitor for sensors reading a temperature higher than a threshold, look for actuators near them, and then control the actuators, for example, to initiate a cooling response in those areas.

B. Data-Centric Application

The second example illustrates how the filesystem abstraction supports a data-centric operation representative of a S&R system. Effective S&R operation requires in-network processing to localize interactions and reduce the size of the data transmitted by the sensors [9]. For example, data from multiple sensors can be aggregated to reduce the overall data size transported to an observer. Or, the data may be analyzed to detect events and initiate responses close to the event location, reducing the cost of data transmission and enhancing response time.

Consider an example where the average temperature in a region (region 10) is periodically reported to a monitoring stationg. We describe the planning and execution of this task from a centralized server perspective for simplicity; however, the namespaces may be maintained, and the task planning carried out, hierarchically and in a distributed fashion

First, the application namespaces are consulted to discover the sensors in that region by using `ls /network/location/region-10/*`. This determines that cluster 1 is within the area of interest. The location information in the namespace is now used to find a set of sensors with the appropriate coverage. In addition, we may consult an energy-based namespace where sensors are categorized in terms of their remaining energy. This allows the application to avoid selecting sensors with low available energy (e.g., S_4 and S_5) leaving only high remaining energy sensors who satisfy the coverage requirements (S_1, S_3, S_6, S_7).

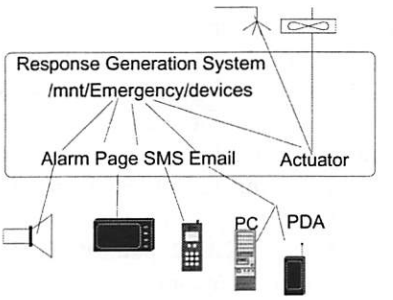


Fig. 3. An S&R system.

Resource namespaces can also support detailed query planning by tracking network-level resources—in our case to determine the routing and aggregating nodes in the network. These namespaces may include sensor connectivity, bandwidth availability, and resource availability. At the end of this step, the task planning is accomplished, and a suitable set of sensors, the dataflow in the network, and in-network processing is determined.

The query is executed as follows. The source sensors are tasked with an appropriate reporting rate (which can later be adapted) to their upstream neighbors as per the determined dataflow path. Basic sensors have support for sending and receiving packets, but some sensors (e.g., cluster heads) support Styx servers and act as multiplexers. Communication between the sensors forming the dataflow is set up using Styx. Application specific in-network processing can be accomplished by customizing packet handlers in these multiplexer nodes. This can be done dynamically (allowing specialized handlers to be moved to appropriate places in the network), statically (at compile time, or within the Styx protocol), or by allowing the application to select among a menu of predetermined handlers.

C. Heterogenous Response System Architecture

In the first example, we demonstrated how we can control actuators embedded with the sensor network to generate the required response. In this example, we describe the flexibility of the proposed framework in terms of incorporating a wide range of heterogenous devices. As an example, consider a S&R system (Figure 3) deployed in a chemical factory to detect any gas leakage. The response generation system takes input from a range of chemical sensors, processes it and then generates the necessary response. The response might include local activities such as controlling actuators embedded within the sensor network or it might include contacting external entities and authorities or both.

If the system is built using the file system abstraction, it might have a directory called /mnt/Emergency and the response generation system might organize different

responses under this directory. For example, upon detecting gas leakage, it might set an alarm to alert local workers and activate the actuators on a sprinkler in order to turn it off. In addition, it might SMS events to medical professionals and e-mail other local authorities.

In many cases, the task force is formed in an ad hoc fashion without any knowledge about underlying sensing infrastructure [10]. With the proposed framework a new device can be mounted on the fly under the /mnt/Emergency directory and the concerned authority can start getting the notification messages immediately. Also, inter-organization communication can be accomplished more easily using simple file commands.

VII. Implementation

We have implemented a prototype which integrates the Styx protocol library with the ns-2 simulator [11]. In the current implementation, during the initialization phase, the cluster head (CH) discovers the neighboring sensors and since the CH is running the Styx file server, it simulates sensing devices as files in a file system hierarchy. In the current implementation, we have incorporated the support for constructing various namespaces within the Styx server. Then the client starts the session with the CH by calling the attach function exposed by the client-side Styx library. The client-side Styx library then encodes this command into a low-level Styx message which is sent over the wireless channel. The Styx server running on the CH interprets this incoming Styx message, processes it, and sends a pointer to its root directory to the client, again using the Styx protocol. It should be noted that, the client-side Styx library exposes a clean file system interface and hides all the low-level details of the Styx protocol from the client. Upon getting the pointer to the root directory, the client is able to navigate this directory structure using the walk command and it reads the files using the read command. In essence, the simulation set-up supports the capability required by the sensor network monitoring example described in Section VI. In addition, with our simulated prototype, we are able to simulate a sensor network consisting of at least few hundred sensors.

We have also developed the basic infrastructure for implementing the file system abstraction on real sensors such as the Stargate and Berkeley motes. We have developed a lightweight file server model suitable for the Motes, which consists of about 1000 lines of code and is less than 8KB in size. Our design incorporates the fact that these Motes have reasonable about of flash memory (a few KB) but much less RAM (few hundred bytes), by extensive use of static structures such as device tables and by judicious use of dynamic memory. We have also adopted the less demanding event-driven model as opposed to using runtime threads. Once this implementation is complete we hope to start using it in problems concerning resource monitoring, calibration, and distributed debugging all leading to more complex data centric applications.

VIII. Conclusion

Sense-and-respond systems are typically heterogeneous and resource-constrained. Under these conditions, system and application development is difficult, especially for domain experts and other developers whose specialty may not be embedded systems. In this paper we have demonstrated how a simple and well-known abstraction, that of a file system, hides much of the underlying complexity, allowing developers to focus on the fundamental challenges of S&R systems. Our initial results with a prototype on the ns-2 simulator suggest that such an abstraction can be practically implemented. Our next step is to port our implementation to a physical WSN such as one constructed from Stargate and Motes.

References

[1] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from Bell Labs," *Computing Systems*, vol. 8, no. 3, pp. 221–254, Summer 1995.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *In Prof. of ASPLOS 2002*.

[3] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Trans. Comput. Syst.*, 1992.

[4] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Record*, 2002.

[5] S. M. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. W. Trickey, and P. Winterbottom, "The inferno operating system," *Bell Labs Technical Journal*, Winter 1997.

[6] R. Pike and D. M. Ritchie, "The styx architecture for distributed systems," *Bell Labs Technical Journal*, 1999.

[7] Y. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC'02)*.

[8] K. Whitehouse and D. Culler, "Calibration as parameter estimation in sensor networks," in *Workshop on Wireless Sensor Networks and Applications (WSNA) 02*, 2002.

[9] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. 5th ACM International Conference on Mobile Computing and Networking (Mobicom)*, 1999.

[10] K. M. Chandy, B. E. Aydemir, E. M. Karpilovsky, and D. M. Zimmerman, "Event webs for crisis management," in *Presented at the 2nd IASTED International Conference on Communications, Internet and Information Technology*, 2003.

[11] "Network Simulator," http://isi.edu/nsnam/ns.

Transversal Issues in Real-Time Sense-and-Respond Systems

Ahmad T. Al-Hammouri
Case Western Reserve University
E-mail: ata5@case.edu

Vincenzo Liberatore
Case Western Reserve University
E-mail: vxl11@case.edu

Huthaifa A. Al-Omari
Case Western Reserve University
E-mail: haa6@case.edu

Stephen M. Phillips
Arizona State University
E-mail: stephen.phillips@asu.edu

Abstract

Networked S&R systems extend human capabilities beyond temporal and spatial barriers with useful applications in broad areas. Involving the physical-world environments, S&R systems must fulfill the intrinsic real-time requirements of these physical environments. However, communication networks lack of QoS can hamper performance and effectiveness of S&R. Therefore, end system strategies must be deployed to retain effectiveness and to enhance performance of S&R. In this paper, we survey transversal issues pertaining to the development of agile S&R systems to work over a geographically scalable network.

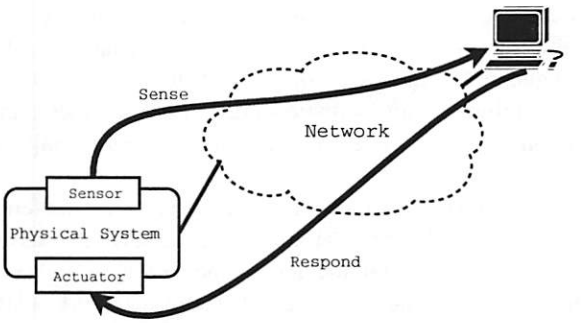


Figure 1: Simplest example of S&R system (with a feedback loop between sense and respond).

1 Introduction

Networked sense-and-respond systems (S&R) extend human reach beyond temporal and spatial barriers. Remote physical environments can then be monitored, controlled, and affected through communication networks (Fig. 1). Examples of potential applications include industrial automation [15], automatic asset management [7], distributed instrumentation [1, 16], disaster recovery [14], unmanned vehicles [11], and home robotics [16].

S&R systems involve a physical environment from which they inherit the real-time characteristics. To ensure operation correctness and to attain maximal performance levels, delivery timelines of sense and respond messages must fulfill the real-time constraints mandated by the specific physical environments. On the other hand, communication networks have inherent non-deterministic behavior, and provide no timeliness or QoS guarantees on data delivery. Consequently, S&R must deploy strategies to alleviate networks' non-determinism and lack of QoS, such as bandwidth limitations, packet losses, delays, and delay jitter. Due to the Internet's end-to-end principle, most of the complexity associated with these strategies must be pushed to end-systems.

Related attempts to solve these problems have focused on control theory, middleware, and databases; whereas networking research has been conspicuously absent in the arena of S&R. Nonetheless, existing methods, such as play-back buffers and congestion control, stress the fact that these problems fall within the scope of networking and thus networks are critical to the resolution of these problems. Since these methods cannot be used the way they are—because they were not developed specific for S&R systems—they must be adapted to work with the S&R systems. This paper presents our conceptual framework for understanding these problems and the open issues that we think are the most pressing and critical for further advancements in the area of S&R. Based on our extensive experiences in the fields of Internet robotics (e.g., [1, 16, 2]), networked control systems (e.g., [5, 8]), industrial automation[17], and COTS middleware platforms (e.g., [2, 11]), we will draw and identify transversal issues related to the development of S&R systems to work effectively over best effort geographically scalable networks.

Heterogeneity of S&R. S&R control environments can differ radically in complexity and in applications. Such environments can range from simple linear systems as in the case of a thermostat to very complex ones, which might include systems of subsystems, as in the case of unmanned autonomous vehicles (UAVs) [11] and in the case of value-chains in manufacturing [9]. Moreover, some systems may include different hierarchical levels of complexity abstraction. For example in UAVs, there are several hierarchical levels. At the lowest level is the direct force level. At this level, the on-board controller issues tasks, such as rotating motors forward or reverse, based on feedback information supplied by sensors (this represents local sense and respond). At the highest level of abstraction are software agents. Software agent carry out high-level tasks, and are responsible of coordinating multiple UAVs into task-oriented teams, which can have impact on different applications, e.g., military transformation [12]. On the other hand, an online auction S&R system would comprise only the software-agent level.

In spite of the striking heterogeneity across different systems' typologies and levels, there are several transversal issues common to the development of agile S&R systems. Transversal issues arise because they address the necessity of providing network QoS for S&R systems to meet their inherent real-time requirements. This paper presents these transversal issues with illustrative examples. These transversal issues are: encapsulation of inner loops (Section 2.1), adaptability and tolerance (Section 2.2), and congestion control (Section 2.3). This paper also highlights differences exist between real-time S&R systems and other real-time applications.

2 Transversal issues

This section discusses the various transversal issues pertaining to real-time S&R systems. It is worth noticing that there are other issues that are critical to distributed systems and applications in general and are not specific to real-time S&R. These issues, such as security, are not covered in this paper.

2.1 Encapsulation of inner loops

An S&R system can include an internal sense-and-respond loop whose actions, i.e., sense or respond, are local to a particular system and are not exchanged over the network, as shown in Fig. 2. This nested sense-and-respond loop establishes a particular form of hierarchical control that highlights the boundary between local and global S&R. Such a configuration often arises in practice. For example, in an *online auction*, a bidder chooses

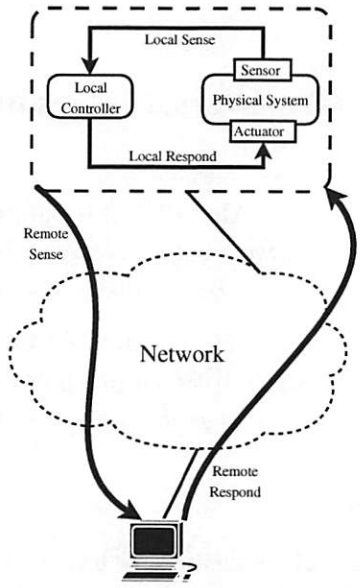


Figure 2: Local and remote sense and respond in S&R.

a maximum price for a particular item (the outer loop). Based on other bidders' behaviors, the auction's agent places bids on this bidder's behalf up to the maximum price (the inner loop). Whenever the bidder's maximum price is outbidded, the auction's agent conveys this to the bidder, who then reacts by choosing a higher bid or by withdrawing from the auction (again, the outer loop). In general, an S&R system in a networked environment will *expose* a certain interface (in the example, the maximum price a bidder is willing to pay for an item), and will *encapsulate* locally other functionality that can include complex decision strategies (in the example, the placing of higher bids based on other bidders reactions). The decision on what to expose globally and what to encapsulate locally is transversal to all applications and taxonomies of S&R and it is critical for the system behavior. In our experience, we have developed the following two general principles to guide the design process regarding encapsulation [1, 11].

In the first place, a system behavior should be controlled locally whenever possible. In many cases, this implies that the remote control, i.e., the outer S&R loop, has access only to the variables that affect the global coordination of the system. Conversely, the remote control would not have access to those variables that are not relevant for distributed control. For example, placing a higher bid on behalf of a bidder is accomplished locally at the auction site, however the maximum bid depends on the willingness of the bidder on how much to pay and so it must be obtained remotely. There are several reasons why S&R loop should be as local as possible. First, the

system's responsiveness and performance are typically superior when a local loop is used. Second, the safety of the system often depends on failback measures that need to be implemented locally to avoid potential failures or unreliable levels of service in the network infrastructure. Finally, a local loop reduces the demands on the communication network. However, some communication is required if cooperative and coordinated action is needed. In general, different variables are relevant in the global setting depending on the application and therefore different variables are exposed or encapsulated depending on the application. Furthermore, power or computational constraints can require certain variables to be remotely controlled even though they could have been otherwise encapsulated locally.

Second, the local control can be often implemented in many different ways and the local implementation can significantly affect the behavior of the globally coordinated system. For example, certain manipulation robots can follow a reference arm location through either *position control* or *force control*. Position control usually results in the robot executing its task quickly. However, a position-controlled robot can apply forces large enough to cause damage to its environment, to the robot, or to both. An alternative to position control is a version of *force control*, whose advantage is that the robot can be compliant with its environment and does not introduce damaging forces as long as certain parameters are appropriately chosen [1]. For example, in the workspace shown in Fig. 3, the ParaDex robot is required to set the position of levers and switches. Such tasks were achieved in force control [1] but position control would have introduced a significant damage in this task space. Force control and position control present a similar interface to a remote controller, but in this application force control was more appropriate for a networked environment in that the robot remains gentle even when connectivity is poor.

2.2 Adaptability and tolerance

To operate effectively over best-effort wide-area networks, S&R systems with tight real-time requirements must be adaptable to network levels of service as expressed, for example, by delays, jitter, packet losses, and bandwidth. A general S&R model is shown in Fig. 1. In the figure, the physical system generates a *sensed* sample i, wraps it inside a packet, and sends it to the other end host, which processes the sensed data and then generates a *respond* message i. The respond message arrives back the physical system after a round-trip delay d_i since when the the sensed sample i was first generated. Due to the nondeterministic nature of communication networks,

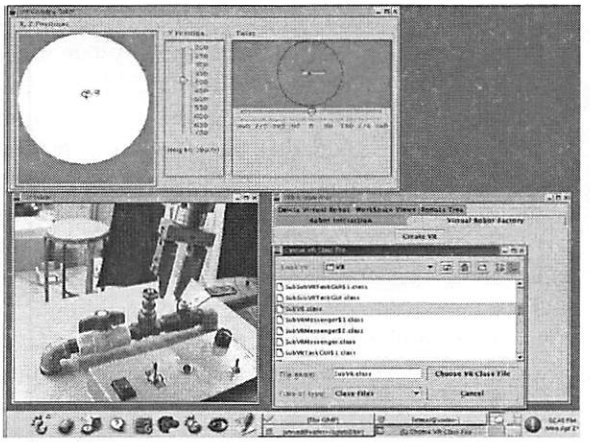


Figure 3: Robot manipulates switches, levers, and valves through force control [1].

delays (d_i's) vary randomly over time. Because of this, we decompose d_i into two components as: $d_i = \tau + \xi_i$, where τ is a nominal round-trip time between the two end hosts and is constant across i's; whereas, ξ_i is the variation, i.e., the *jitter*, in the round-trip delay i from the nominal value. This decomposition allows us to partition the problem of time-varying delays into two subproblems: fixed delays and jitter. Apart from delays and jitter, either sense or respond packets may be dropped out from the network. In the subsections to follow, we thus discuss methods to address these three issues, i.e., fixed delays, jitter, and packet losses, while bandwidth allocation is deferred to Section 2.3.

2.2.1 Round-trip delay

S&R systems could involve systems with inherent physical dynamics that require upper limits on delays. Consequently, long delays degrade performance and may result in S&R systems' misbehavior. In the online auction example of Section 2.1, an item's current bidded price and participants' bids must be conveyed in a timely manner and must not be delayed longer (say, after the auction's time elapses) such that the auction proceeds in an efficient way.

When delays are fixed, i.e., $\xi_i = 0$, several methods can be deployed to ameliorate the adverse effect of delays. One method is by sacrificing performance to preserve effectiveness. For instance, when teleoperating a robot in presence of transmission delays, an operator would issue commands in smaller steps, for example, the operator would command a robot to move only a couple of inches, then he would hold back waiting to see the visual feedback before issuing another command, and so

on. A second method is to use observers. An *observer* is an approximate model of the physical system used to simulate system's dynamics. Based on the latest sensed information, the observer extrapolates the system state to predict a future state after one round-trip time, and then the respond signal is issued accordingly. Not all system dynamics can be captured and predicted by observers because some systems have complex dynamics and/or complex interactions with their environments, one such example being the online auction. A third method is using encapsulation in that the system parts that necessitate short delays are implemented locally—if this is possible, see Section 2.1.

2.2.2 Delay jitter

Jitter leads to unpredictable delivery-times of sense and respond messages. Whereas some S&R are not sensitive to jitter as long as round-trip delays are within some range, others are. Examples of the first is the online auction; and of the second is networked control systems [18]. In addition, jitter causes inaccurate predictions in those systems that use delay-compensation techniques based on observers (see Section 2.2.1). Play-back buffers can be deployed at the physical system side to smooth jitter and to apply control signals at predictable times, i.e., ensuring the situation of $\xi_i = 0$ and thus $d_i = \tau$. Play-back buffers have been used in multimedia streaming [10]. However, in multimedia streaming, jitter is addressed in terms of one-way delays, and play-back delays are determined by the end-system that receives the stream. In S&R, on the other hand, the situation is different. First, jitter in the round-trip delays is the concern. Second, play-back delays are determined by the controller, which is away from the host that plays back (i.e., applies) the signal. Furthermore, multimedia and S&R possess different performance metrics. Therefore, methods and algorithms used in multimedia play-back must be adapted or changed to work for the case of S&R.

The idea of play-back buffers is that packets are delayed by the receiver a certain amount of time, called play-back delay, before being processed. Packets are processed only if they arrive before the scheduled processing times; otherwise, i.e., a packet arrives after the scheduled time, it is dropped and is considered lost. Therefore, choosing an appropriate value for the play-back delay should compromise between delay (see Section 2.2.1) and losses (see Section 2.2.3).

2.2.3 Packet losses

S&R traffic carries vital data and hence packet losses are undesirable. Although packet losses can be minimized with methods like *Forward Error Correction* [10], they cannot be entirely eliminated. Retransmission of lost packets is an inappropriate solution, either. This is true because by the time a packet has been discovered to be lost and retransmitted, the system would have evolved to a newer state—and thus the retransmitted packet would have been based on stale information.

When a packet carrying either a sense or a respond information is lost, then the system behavior is equivalent to the case when the corresponding sensed sample was not generated at all. Effectiveness of an S&R system can still be attained if subsequent packets arrive intact and that the rate of generating and sending sensed data is higher than the rate of variation of the system's state. Let us illustrate with an example. A thermostat is installed to measure the temperature of an environment. The temperature varies as 1^o per minute. Suppose the whole system is sensitive to variations of 0.1^o and the temperature measurements are made 50 times a minute. In every five consecutive packets, if four are dropped out from the network, the system will continue to function properly.

Oversampling, which is sensing at a rate higher than what is actually needed, makes S&R systems more tolerant to packet losses. However, two important issues need to be addressed when oversampling. First, oversampling increases demand on the bandwidth and may cause congestion, which in turn leads to more packet delays and losses. Second, there must be a criterion to ascertain how much to oversample. In Section 2.3, we address both of theses issues.

2.3 Congestion control

The introduction of congestion control into TCP solved the problem of congestion collapses that were occurring during 1980s. Congestion control was one of the reasons that the Internet scaled up to its size today. Due to the original philosophy of Internet—the end-to-end principle—the entire implementation of the congestion control scheme was delegated to end-systems, which are senders and receivers.

The main objective of congestion control is to match senders' transmission rates to network capacity. Congestion control is transversal among all the levels discussed in Section 1. At the agent level, agents are applications built atop the transport layer and in most cases the conventional TCP congestion control is involved when agents move from a place to another or when they exchange high-level messages. Therefore, we will not cover it here. For lower level layers, congestion control is used to provide a *fair* bandwidth allocation:

- Between S&R real-time traffic and other non-real-time traffic, and

- Among different S&R traffic.

We emphasize that *fair*, here, should not be mistakenly understood to mean even or equal; rather, it means that allocation is related to intrinsic bandwidth requirements and is affected by the environment to be controlled. Allocating bandwidth between S&R and other non-real time traffic is necessary because S&R traffic is less elastic than other traffic and it requires minimum bandwidth guarantees. Allocating bandwidth among different S&R traffic is necessary because different physical environments differ in speed of physical dynamics. As a result, how frequent sense-and-respond messages must be exchanged—and thus the bandwidth allocation—differ from one system to another. We elaborate on this with an example.

Assume a hypothetical scenario where two S&R's used to control the process of filling two irrigation tanks with water up to a certain level. Assume further the two tanks have the same capacity but being supplied with two different water pipes: one supplies water at rate 20 liters per minute and the other supplies at rate 180 liters per minute. The level of the second tank needs to be sensed at least 180 times per minute so that the error in the water level may not exceed 1 liter. On the other hand, it is sufficient for the first tank to be sensed at least 20 times per minute so that not to exceed an error of 1 liter. If the two S&R's share one network link that can transmit data at a maximum rate of 300 sensed samples per minute, the question then becomes: how to divide the bandwidth among these two systems in an efficient way and to obey the aforementioned requirements. One feasible solution is to give each system its minimum requirements, i.e., 20 samples per minute and 180 samples per minute for the first and the second respectively, and then to divide the remaining unused bandwidth in anyway between the two. Another natural and better way is to divide the bandwidth proportional to the speed of water flow, i.e., 30 samples per minute and 270 samples per minute for the first and the second systems respectively.

This particular example is easy because of two reasons. First, the environment is static, that is, we have fixed number of systems and fixed network parameters. Second, the dynamics of each system, i.e., the rate of water flow, is globally known. In practice, however, systems use and relinquish network resources in a dynamic manner and one system may not know dynamics of others. In [4], an admission-control protocol is proposed to solve these issues. Systems need to register with a master node before gaining admission to the network. The bandwidth is time-divided among active systems, that is, each systems is given specific time slots during which it can send or receive data. The disadvantages of this approach are that it is fully centralized and it requires clock synchronization among all entities in the network.

Our contribution in this area is to devise a distributed approach that is scalable, flexible and reconfigurable. We measure the performance of each S&R system, i, by a performance function, $U(A_i, w_i)$. A_i captures the physical dynamics of system i, and w_i is the bandwidth allocated to system i. Bandwidth w_i can be thought of as the rate at which a S&R system can generate and transmit sensed samples. We mathematically formulate the overall objective as follows:

$$\begin{aligned} \max \quad & \sum_i U(A_i, w_i), \\ \text{s. t.} \quad & \sum_{i \in S(l)} w_i \leq C_l, l = 1, \ldots, L \end{aligned} \tag{1}$$

where $S(l)$ is the set of systems whose end-to-end flow paths use link l, B_l is the capacity of link l, and L is the total number of links in the network. In other words, the objective is to allocate the network links capacities in such a way to maximize the aggregate performance of all the systems. We use the results and the techniques published in [13] to achieve the objective expressed in (1) in a distributed manner. Specifically, the authors in [13] formulated the problem as a convex optimization problem and they used the Lagrange multipliers method to decompose the problem into separable subproblems. Based on their approach, each link computes a so-called *price* variable, which measures the mismatch between aggregate flow rates and the capacity of the link, i.e., the congestion at the link. Links report prices back to end-systems whereby end-systems adjust their sending rates (i.e., the frequency of generating and sending sense and respond messages) appropriately. For the solution to converge to an equilibrium, the performance function must be concave, monotonically increasing, and twice continuously differentiable; and its second derivative must be greater than zero. The advantage of this approach can be manifested in two scenarios. First, when there are few S&R systems competing for the network, this approach allocates the bandwidth in a way that every system operates with high performance. Second, when there are too many S&R systems, the network bandwidth may not be sufficient to ensure high performance for each system. However, the approach allocates the bandwidth in order to achieve the highest aggregate performance for all S&R systems.

In [3], we demonstrated the applicability of this approach for the class of scalar linear control systems whose system dynamics evolve according to the following differential equation:

$$\dot{x}(t) = ax(t) + u(t) \tag{2}$$

where $x(t)$ is the system state; a is the system constant—a is larger for faster system dynamics; $u(t) = -kx(t_j)$ is the *respond* signal, which is simply a constant, k, multiplied by the last *sensed* signal.

A representative performance function would take the form of:

$$U(A_i, w_i) = \frac{a_i - k_i}{a_i} e^{\frac{a_i}{w_i}} \quad (3)$$

This definition is based on the error, derived in [6], that the system develops when using bandwidth equal to w_i. Note that $U(A_i, w_i)$ in (3) satisfies all conditions mentioned before.

3 Conclusion

In this paper, we have discussed fundamental issues for designing effective real-time S&R systems over communication networks. Such issues stem from two facts. First, communication networks are typically best-effort media with no QoS provisioning guarantees. Second, operation correctness and performance of real-time S&R systems depend on their ability to adapt to networks levels of service. Thus, intelligent end-system strategies must be deployed to conceal the adverse effects of networks' lack of QoS on the effectiveness of real-time S&R systems.

Acknowledgments

We thank Michael S. Branicky and Nathan Wedge for helpful conversations. This Work has been supported in part under NSF grant number CCR-0329910, Department of Commerce grant number TOP 39-60-04003, and NASA contract number NNC05CB20C.

References

[1] A. Al-Hammouri, A. Covitch, D. Rosas, M. Kose, W. S. Newman, and V. Liberatore. Compliant control and software agents for internet robotics. In *Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, 2003.

[2] Ahmad T. Al-Hammouri. A distributed framework to facilitate human-robot remote interaction. Master's thesis, Case Western Reserve Universiy, Cleveland, Ohio, Januray 2004. Advisor: Vincenzo Liberatore.

[3] Ahmad T. Al-Hammouri and Vincenzo Liberatore. Optimization congestion control for networked control systems. In *IEEE Infocom Student Workshop*, Miami, FL, March 2005. Abstract.

[4] L. Almeida, J.A. Fonseca, and P. Fonseca. A flexible time-triggered communication system based on the controller area network: Experimental results. *FeT'99, Int. Symposium on Fieldbus Technology*, September 1999.

[5] M. S. Branicky, V. Liberatore, and S. Phillips. Co-simulation for co-design of networked control systems. In *American Control Conference*, 2003.

[6] Michael S. Branicky, Stephen M. Phillips, and Wei Zhang. Scheduling and feedback co-design for networked control systems. In *Proc. IEEE Conf. on Decision and Control*, Las Vegas, December 2002.

[7] Alexander Brewer, Nancy Sloan, and Thomas L. Landers. Intelligent tracking in manufacturing. *Journal of Intelligent Manufacturing*, 10(3-4):245–250, September 1999.

[8] Justin R. Hartman, Michael S. Branicky, and Vincenzo Liberatore. Time-dependent dynamics in networked sensing and control. In *American Control Conference*, Portland, OR, June 2005.

[9] Albert Jones and Abhijit Deshmukh. Test beds for complex systems. *Commun. ACM*, 48(5):45–50, 2005.

[10] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley Longman, Inc., 2001.

[11] V. Liberatore et al. IP communication and distributed agents for unmanned autonomous vehicles. In *AIAA-UAV*, 2003.

[12] Grace Y. Lin and Jr. Robert E. Luby. Transforming the military through sense and respond, January 2005. IBM White paper.

[13] Steven H. Low and David E. Lapsley. Optimization flow control—I:basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[14] L. Matthies et al. A portable, autonomous, urban reconnaissance robot. In *The 6th International Conference on Intelligent Autonomous Systems*, July 2000.

[15] W. S. Newman et al. Design lessons for building agile manufacturing systems. *IEEE Trans. Robotics and Automation*, 16(3):228–238, 2000.

[16] Marco L. Ngai, Vincenzo Liberatore, and Wyatt S. Newman. An experiment in remote robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2190–2195, 2002.

[17] B. P. Robinson and V. Liberatore. On the impact of bursty cross-traffic on distributed real-time process control. In *Workshop on Factory Communication Systems (WFCS)*, 2004.

[18] Z. Wei, M. Branicky, and S. Phillips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99, February 2001.

M-ECho: A Middleware for Morphable Data-Streaming in Pervasive Systems

Himanshu Raj Karsten Schwan
Ripal Nathuji
Center for Experimental Research in Computer Systems
Georgia Institute of Technology, Atlanta, GA 30332
{*rhim, schwan*}*@cc.gatech.edu, rnathuji@ece.gatech.edu*

Abstract

The end-to-end performance of pervasive mobile systems is commonly dictated by the availability of resources at the *weakest link*. However, a number of runtime adaptations or *morphing* steps can be performed to tune the system performance. In this paper, we present M-ECho, a middleware for system *morphing*. M-ECho is designed with focus on data streaming applications, specifically in the field of pervasive mobile systems. We consider an autonomous robotics application comprising of a set of cooperating mobile robots to demonstrate and evaluate M-ECho's system morphing capabilities. Optimizations are based upon metrics of average instantaneous power consumption at a single node (local) as well as the power consumed by all participants (global). Experimental results show that M-ECho is able to achieve improved end-to-end performance with its dynamic *code morphing* techniques.

1 Introduction

Pervasive mobile systems are often comprised of resource limited mobile nodes. These nodes can be characterized by decreased computing and communication capabilities, as well as limited battery power. Due to continuous environmental changes and changing system objectives, it is required that these systems adapt themselves in order to optimize their resource usage and performance.

In this paper, we present M-ECho, a middleware that provides support for continuous system adaptation and evolution, termed *system morphing*. In particular, M-ECho focuses on data streaming applications in pervasive domain. Specifically, the middleware (1) provides mechanisms for runtime behavioral changes and (re)deployment of program components and (2) makes middleware, and thereby applications, *aware* of current resources. Resource awareness involves runtime interactions between middleware and the underlying distributed platform. *Code morphing* is the runtime alteration of the implementations of specific program components, the goal being to dynamically create components with the properties that best match the system's resource usage directives. Code morphing is carried out by actions that include runtime code generation, code deployment or re-deployment, and by *change transactions* [8] that guarantee desired safety properties when distributed programs are changed at runtime. Morphing is triggered by changes in program objectives or behavior and/or in currently available resources.

Apart from traditional end-to-end performance criterion, such as increasing achievable bandwidth and reducing latency, efficient energy usage is an important objective for pervasive mobile systems since it directly relates to the longevity of the system. In this work, we focus on this criteria for evaluating the morphing capabilities of M-ECho for target systems.

Past research on energy efficiency in mobile systems has typically sought to prolong the battery lives of individual devices. Recent results extend to entire systems. An example is message routing in ad-hoc networks, where cooperative routing is performed to route messages to avoid using power-poor devices, thereby permitting the entire system to continue its operation [4]. Our work addresses the *system-wide power* consumed by the processing and communication actions of a distributed application. We focus on power and not on energy since (1) we assume that our tasks are long running and (2) available energy (battery power) is not a monotonically decreasing resource in the system. There might be opportunities for charging batteries in future or change in the total number of nodes comprising the system. The premise is that the application can function only as long as the *weakest* (the power poorest) device hosting its components. There is a variety of techniques to deal with predictable device failures due to power paucity, including migrating functionality par-

tially or completely onto different devices. Instead of requiring an application to explicitly implement such techniques, our research is creating middleware solutions that make it easier for developers to implement such techniques or even to automate their use.

The key advantage of code morphing over per device or per subsystem techniques for managing power is the ability to integrate application-level changes with system-level behavior changes. To evaluate and quantify this advantage, we have implemented code morphing in the M-ECho middleware. M-ECho implements a publish/subscribe paradigm of inter-machine communication, mapping the logical channels to link cooperating devices for inter-device communication. The data traversing these channels is described as events, where event providers, consumers, or intermediates can operate on events using well-defined event handlers. Event handlers are the software components for which we implement code morphing. Such handler morphing differs from prior work in application adaptation due to its ability to re-deploy handlers on demand and as currently needed by the application or its execution environment. The redeployment uses either dynamic compilation and code generation or static code repository. In a sense, handler morphing combines the abilities of compilers to generate the code most suitable for a platform (and its current resources) with parameterization or system-based techniques for dynamic program adaptation.

While M-ECho's implementation of code morphing targets event handlers, the idea of code morphing generalizes to other systems. Class and agent based migration has been used in Java and Corba frameworks [9, 7]. Tempo [14] uses code parameterization and compiler assisted code specialization for runtime optimization.

This paper makes two key contributions.

- It describes the code morphing mechanisms provided by M-ECho, where event handlers are dynamically modified in order to achieve desired changes in application behavior in response to system-wide changes in energy resources.
- Experimentation with a realistic, distributed application in the autonomous robotics domain both motivates this work and provides insights into the utility and limitations of middleware-level techniques like code morphing to improve performance. In this application, a team of autonomous robots must communicate environment information like images or laser data to each other, in order to attain some joint goal, such as foraging for resources. Attaining such a goal requires cooperation and joint actions, including tasks like motion and path planning.

An interesting insight derived from the use of M-ECho with the robotics application is a differentiation of *node local* from *system global* instantaneous power consumption as target performance metric. This is particularly relevant in mobile applications in which there are future opportunities for re-charging batteries (e.g., consider robots docking at a re-charging station). An interesting example in the robotics application is the use of compression techniques for image sensor data. JPEG compression, for instance, can reduce the instantaneous power consumption of the overall system by approximately $\sim 5\%$, even when we keep the full quality. If we reduce the quality to 75%, the overall system power consumption reduces to $\sim 8\%$. However, if the data provider (i.e., server) has the opportunity to re-charge its batteries in the near future, whereas the client does not, then it may be better to use simpler data reduction techniques, such as an 8 bpp (bits per pixel) grayscale conversion of a 24 bpp color image. This reduces client's power consumption up to $\sim 17\%$, at the cost of increasing server's power consumption by $\sim 10\%$. This tradeoff is entirely reasonable when server vs. client initial energy levels differ (and both should run out of power at approximately the same time) and/or when the server has future opportunities to re-charge whereas the client does not.

2 Issues with Power Consumption in Autonomous Robotics

Energy conservation has always been a goal for mobile devices due to their limited battery power. In distributed mobile systems, a range of techniques is used to conserve system-wide energy and ensure application longevity, ranging from node-level methods like dynamic voltage scaling [17], periodic sleeping or frequency scaling to reduce idle times, to compiler techniques such as generating energy-efficient code [11], to OS specific methods like power-aware scheduling [20], to application-level methods like agent migration [9] and changing attributes such as fidelity [6]. Amongst such techniques, online methods for power management typically address specific subsystems, beginning with early work on power-aware communication [10], to recent work on power-aware routing in wireless ad-hoc networks [18]. Such methods can take advantage of the fact that participating nodes tend to be freely interchangeable, so that one node can transparently replace another. In comparison, in the robotics applications addressed by our work, nodes have certain roles, which must be taken into account explicitly when applications are adapted.

Our target application domain is distributed autonomous robotics. Our specific application is one in which a team of robots cooperated to execute some common task, an example being disaster recovery and relief. Each rescue robot can perform multiple actions (i.e., play

different roles), constrained by the availability of certain peripherals. An example is a robot equipped with a laser but not with odometry capabilities, which means that the robot can perform environment sensing but not localization (i.e., help the team determine its robots' respective positions). Other constraints are due to current application or environment state. A robot with a good view of a disaster site, for instance, may have to maintain its critically important role as a sentry, despite its ability to also perform other tasks. Another interesting characteristic of this application is that energy need not be a uniformly decreasing resource. A robot may be able to increase its energy resources due to the availability of an external source of energy. The robot may be able to move towards an external power source and charge its batteries without causing a disruption of service in the system. Or, a robot may have solar batteries, which can be re-charged when the robot enters a sunny environment.

The specific prototype application being constructed by collaborating robotics researchers [1] will consist of up to 100 mobile robots equipped with diverse sensors such as laser, sonar and cameras, 802.11b-based wireless communication device(s) and Intel XScale based CPUs. This paper focuses on the robot team's IT infrastructure, in anticipation of future micro-robots and current micro-sensor-based systems where power consumption is dominated by the IT infrastructure's actions and behavior. The experiments conducted in this paper implement a sample collaborative task, using video streams to emulate sensor communications across different robots and power behavior of a localization algorithm.

The following list concisely articulates the energy-related goals of code morphing pursued for our mobile application. The objective is to extend the application's longevity.

1. Maintain suitable per device power health by modifying application behavior subject to current robot roles, device state, and application state. The goal is to always run application codes where most appropriate for the entire application, thereby optimizing application-wide metrics of utility/power. A sample method is to offload (i.e., delegate) tasks to the most appropriate robots, based on their current power health, capabilities, and application state.

2. Reducing application communication overheads, the method being to dynamically eliminate unnecessary data from event communications, thereby reducing power usage. This implies placing appropriate code (i.e., M-ECho event handlers and filters) onto data-providers (e.g., robots with sensors).

3. Create application-level overlays to help with task offloading, where codes ordinarily running on single nodes are morphed to processing pipelines mapped across multiple, network-connected machines. In comparison to prior work on network-level message routing, our work creates middleware overlays that route and process events, matching robots' power levels and processing capabilities. This is akin to assigning roles to sensor nodes in a wireless sensor network [16].

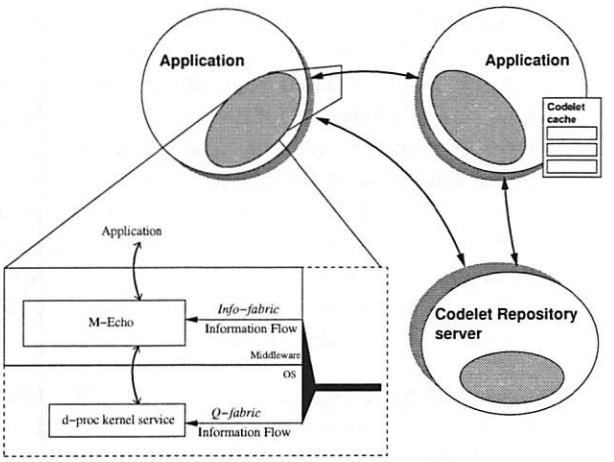


Figure 1: M-ECho Architecture

3 System Morphing with M-ECho

3.1 M-ECho Software Architecture

The M-ECho framework (refer to figure 1) consists of the following components:

1. *InfoFabric Middleware Platform.* The InfoFabric middleware platform is based on the ECho [5] publish/subscribe middleware which presents a distributed source/sink view for the whole system. Information is exchanged between source and sink nodes in form of *events*. *Event handlers* are deployed at the sink node to respond to specific events, while *filters* are deployed at either source or sink nodes to perform information filtering. These event handlers and filters are application specific. The middleware platform also interacts with kernel-level d-proc monitoring service which is part of Q-Fabric [15] infrastructure to receive information about system level events such as the remaining battery power at a node. Infofabric enables data aggregation at nodes, much in the same way provided by DFuse [12] and TAG [13]. The key difference is that the unit of aggregation are ECho events vs. raw sensor data.

   Since M-ECho utilizes the publish/subscribe paradigm of communication, a straightforward

representation of morphable components are the handlers and filters applied to the data events that traverse M-ECho-based applications. In fact, M-ECho's implementation of filters and handlers (also termed as code segments or codelets hereafter) simplifies morphing by limiting the generality of code and state compared to that of arbitrary program components. The codelets are either compiled dynamically using dynamic code generation or a suitable binary version is obtained from the *codelet repository service*, described below.

2. *Codelet Repository Service (CRS).* The CRS functions as a repository for codelets. Codelets are stored along with multiple attributes. Examples of these attributes include the format such as binary vs. ECL source (ECL is a subset of C language [5], OS/platform and a cryptographic signature hash. The service provides a traditional FTP like get/put interface for obtaining/publishing codelets. By storing some pre-compiled versions of a codelet on the repository, we can save on the cost of dynamic code generation wherever possible.

3.2 Code Morphing Techniques

Our morphing work takes an overall system approach to dynamic adaptation. In other words, based upon system requirements (these requirements may be local or global), we adapt the application components (including middleware) and the environment components (such as OS and network). In the presented work, we focus on morphing in M-ECho via adaptation of application components.

Following are the specific morphing techniques that M-ECho currently implements:

- *Code parameterization.* We modify particular parameters to affect the outcome of a certain *codelet* such as changing the number of items that a aggregating function might be using, changing the quality parameter that dictates the computation being performed by a codelet and changing the rate of events to be generated.

- *Code substitution.* We replace one codelet by another. This is assisted via dynamic code generation for the specific platform or via CRS, as described above.

- *Code migration.* We use dynamic (re)partitioning and (re)deployment of application components based on environmental attributes provided by the Q-Fabric infrastructure such as available network bandwidth and remaining node power.

We collectively refer to the morphing techniques used by M-ECho as code morphing. The current version of M-ECho supports all the code morphing techniques described above. As we show later, with code morphing alone we are able to achieve goals 1 and 2 as stated in section 2.

4 Experimental Results

Our experimental scenarios are based upon a set of robots exchanging sensor information in order to perform localization and path planning. Power measurements focus on typical robotic computing subsystems. In particular, we use PXA255 XScale CPU based evaluation boards and StrongARM CPU based iPAQs to mimic the computing system of a robot. Power measurements are obtained by monitoring current with a measurement circuit based upon a current sensor IC, and simultaneously measuring voltage. These values are measured with the Picoscope 212/50 PC-oscilloscope [2]. In the applied mode, the oscilloscope returns an average of a set of samples obtained at 50 million samples per second every 1ms to measure single shot signals. We report the average of these power values as *average instantaneous power.*

We enhanced the commonly used Player/Stage robotics framework [3] to use M-ECho middleware. The current implementation of Player/Stage uses parameterization to perform limited morphing, adapting event rate(s) and event type. Using M-ECho, we are able to enhance the capabilities of the framework (1) to perform codelet substitution and (2) to perform codelet (task) migration across robots, all of which directly affect power usage. The goal is to improve energy usage for all devices participating in the application, that is, to extend the ability of all devices to continue to pursue their joint application-level tasks. In this context, we look at two energy-intensive tasks of this application, the acquisition and exchange of sensor data and the task of localization.

In our experimentation, former is implemented as the exchange and processing of video images captured by device-mounted cameras. Code morphing techniques such as code parameterization and code substitution are used to utilize diverse data encoding and compression techniques in order to control energy usage at information providers vs. consumers. For the latter task, we use a localization application which is part of the enhanced player/stage framework. The application uses the adaptive monte-carlo algorithm [19] and continuously tries to localize the robot based on the sensor data. We use a log of laser readings obtained from one random walk of the robot in a synthetic environment. The localization application reads sensor data from this log and computes the localization information. We use code migration to show that it is possible to achieve power savings along with

better QoS when localization is run locally on the robot vs. off-board.

4.1 Code Parameterization and Substitution

To simulate sensor date exchange between robots, we use a media application running on iPAQs, in which a source sends out raw 24-bit color 352X240 PPM camera images over a 802.11b wireless network and a client (or sink) displays them. The application utilizes several data reduction methods for system morphing. These methods include JPEG compression and image transcoding such as grayscale (GRAY) and black&white (BW) conversion and image cropping (CROP). System morphing techniques used are code parameterization, such as changing the JPEG compression level, and code substitution, such as changing data reduction methods (which requires changing the filters and handlers at source and sink respectively).

Table 1 shows the average instantaneous power consumed (in watts) by the a source and a sink node. *Idle* shows the base case power consumption for both nodes when they are idle. When the application is running, source sends data as fast as it can, assuming that it always has information to communicate. The table lists power values for different data reduction methods. Numeric value adjacent to JPEG denotes the quality parameter value used by the JPEG compression. The frame rate observed by the sink and the normalized power consumption for the desired rate of 1 frame per second (refer to the discussion below) are also shown.

We compare the different methods as following. Given a method can achieve more QoS (in our case, frame rate) than desired, we compute the normalized power as following:

$$Norm\ power(i) \quad = \quad F/T * 1/r_i * (P_i - I) + I$$

where r_i is the QoS (achieved frame rate) and P_i is the average instantaneous power for method i, I is the idle power and F/T is the desired frame rate. To obtain system power, we add respective power values for source and sink nodes. If the frame rate achievable is less than desired, the normalized power at a node is P_i (as is the case for BW and GRAY). However this implies a lower utility for the application (we assume that utility is a smooth monotonically decreasing function and not a unit step function).

These experiments demonstrate that the choice of morphing technique and corresponding method depends on the objective. For a desired QoS of 1 frame per second for our application, if the objective is to optimize client average power usage (regardless of the utility), morphing system can use code substitution to employ encoding methods such as BW or GRAY. If the objective is to optimize global average power usage, then code substitution can be used to employ JPEG compression. Furthermore, if we can tolerate some image quality loss, code parameterization can be employed to use 75% quality for optimal overall average system power usage.

CPU	Local		Remote	
Freq (MHz)	P (W)	T (s)	P (W)	T (s)
400	4.24	39	3.37	38
300	3.75	59	3.2	38
200	3.62	78	3.16	38

Table 2: Average instantaneous power usage and run time for a localization application on a node where localization algorithm is run locally vs. on a remote node

4.2 Code Migration

A common approach to reducing local power consumption at a node is to take advantage of CPU frequency/voltage scaling [17]. However, a reduced frequency often results in an impact to QoS. For example, for a localization application running on the evaluation board, table 2 shows that the local average instantaneous power consumption (Local P (W)) decreases with lower frequency (the processor automatically scales to the lowest voltage to support the desired frequency). However, the time (T) required to compute localization increases, thereby nullifying any gains obtained from frequency scaling. For our application, this effect is magnified due to the architectural characteristics of the PXA255 CPU. The localization algorithm requires floating point operations which are not supported in hardware. For all FP operations, the hardware traps to the OS and software libraries are used to perform the actual operations. This overhead can dominate performance for floating point intensive applications.

Assuming that the environment consists of some power and performance rich nodes that have the ability to periodically recharge themselves (i.e. their power consumption is not of concern in the performance of the system) and can process data faster than the local node, we demonstrate that the localization application can directly benefit from code migration. Table 2 shows the local node's average instantaneous power consumption when the localization algorithm is run remotely (Remote P (W)). In this case, local node not only uses less power with lower frequency, it can still finish the desired task in almost the same time as with higher frequencies. This also serves as an example of system morphing due to restricted node capabilities.

Method	Inst. Power (W)		Frame Rate	Normalized power (W)	
	Sink	Source	(Sink)	Sink	Source
Idle	1.66	1.49	0	1.66	1.49
Nocomp	2.16	1.96	1.9200	1.92	1.73
JPEG 100	2.3	2.12	3.9000	1.82	1.65
JPEG 5	2.32	1.91	6.6250	1.76	1.55
JPEG 75	2.33	1.97	6.1000	1.77	1.57
BW	1.59	2.01	0.3797	1.59	2.01
GRAY	1.63	1.94	0.3750	1.63	1.94
CROP	2.14	2.05	14.1375	1.69	1.53

Table 1: Source and sink power consumption for parameterization and substitution code morphing techniques

5 Conclusions and Future Work

In this paper, we present the design of M-ECho and its code morphing capabilities. Using M-ECho for an autonomous robotics application in a pervasive mobile domain, we demonstrate power performance benefits of code morphing techniques.

Our future efforts are focused on:

- Using compiler assisted techniques for the dynamic compilation of codelets for achieving power benefits, such as mixed code generation [11].
- Quantifying the energy cost of dynamic compilation of codelets vs. the use of CRS.
- Building morphable application level overlays based on node's capabilities to achieve goal 3 as stated in section 2.

6 Acknowledgments

We are thankful to Keith J. O'Hara and other robotics collaborators at the Borg Lab [1]. This research was supported in part by a NSF ITR award.

References

[1] The borg lab. http://www.cc.gatech.edu/~borg.

[2] Picoscope. http://www.picoscope.com/.

[3] The Player/Stage Project. http://playerstage.sourceforge.net/.

[4] CHOKHAWALA, J., AND CHENG, A. M. K. Optimizing Power Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of WIP Session, Real-Time and Embedded Technology and Applications Symposium* (Toronto, Canada, 2004).

[5] EISENHAUER, G., BUSTAMANTE, F., AND SCHWAN, K. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing* (2000).

[6] FLINN, J., AND SATYANARAYANAN, M. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Transactions on Computer Systems* (May 2004).

[7] GU, X., NAHRSTEDT, K., MESSER, A., GREENBERG, I., AND MILOJICIC, D. Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environment. In *Proc. of IEEE International Conference on Pervasive Computing and Communications* (2003).

[8] ISERT, C., AND SCHWAN, K. ACDS: Adapting Computational Data Streams for High Performance. In *Proceedings of International Parallel and Distributed Processing Symposium* (2000).

[9] KON, F., YAMANE, T., HESS, C., CAMPBELL, R., AND MICKUNAS, M. D. Dynamic Resource Management and Automatic Configuration of Distributed Component Services. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems* (2001).

[10] KRAVETS, R., AND KRISHNAN, P. Power Management Techniques for Mobile Communication. In *Proceedings of MOBICOM* (1998).

[11] KRISHNASWAMY, A., AND GUPTA, R. Profile Guided Selection of ARM and Thumb Instructions. In *Proceedings of ACM SIGPLAN Joint Conference on Languages Compilers and Tools for Embedded Systems* (2003).

[12] KUMAR, R., WOLENETZ, M., AGARWALLAA, B., SHIN, J., HUTTO, P., AND RAMACHANDRAN, U. DFuse: A Framework for Distributed Data Fusion. In *Proceedings of SenSys* (2003).

[13] MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *Proceedings of OSDI* (2002).

[14] NOEL, F., HORNOF, L., CONSEL, C., AND LAWALL, J. L. Automatic, Template-Based Run-Time Specialization: Implementation and Experimental Study. In *Proceedings of the International Conference on Computer Languages* (1998), IEEE Computer Society, p. 132.

[15] POELLABAUER, C. *Q-Fabric: System Support for Continuous Online Quality Management.* PhD thesis, College of Computing, Georgia Institute of Technology, 2004.

[16] ROMER, K., FRANK, C., MARRON, P. M., AND BECKER, C. Generic Role Assignment for Wireless Sensor Networks. In *Proceedings of SIGOPS European Workshop* (2004).

[17] SIMUNIC, T., BENINI, L., ACQUAVIVA, A., GLYNN, P. W., AND MICHELI, G. D. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proceedings of Design Automation Conference* (2001).

[18] STOJMENOVIC, I., AND LIN, X. Power Aware Localized Routing in Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems* (November 2001).

[19] THRUN, S., FOX, D., AND BURGARD, W. Monte Carlo Localization With Mixture Proposal Distribution. In *Proceedings of the National Conference on Artificial Intelligence* (2000).

[20] YUAN, W., AND NAHRSTEDT, K. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of SOSP* (2003).

The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems*

Amol Bakshi*, Viktor K. Prasanna*
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089 USA
{*amol, prasanna*}*@usc.edu*

Jim Reich, Daniel Larner
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
{*jreich, larner*}*@parc.com*

Abstract

The Abstract Task Graph (ATaG) is a data driven programming model for end-to-end application development on networked sensor systems. An ATaG program is a system-level, architecture-independent specification of the application functionality. The application is modeled as a set of abstract tasks that represent types of information processing functions in the system, and a set of abstract data items that represent types of information exchanged between abstract tasks. Input and output relationships between abstract tasks and data items are explicitly indicated as channels. Each abstract task is associated with user-provided code that implements the actual information processing functions in the system. Appropriate numbers and types of tasks can then be instantiated at compile-time or run-time to match the actual hardware and network configuration, with each node incorporating the user-provided code, automatically generated glue code, and a runtime engine that manages all coordination and communication in the network. This paper primarily deals with the key concepts of ATaG and the program syntax and semantics. The end-to-end application development methodology is discussed briefly.

1 Introduction

Wireless sensor networks allow embedded, dense monitoring of the physical environment. The challenge in programming a sensor network is to coordinate the sensing, collaborative processing, and data flow in the network *correctly*, so that the desired functionality is achieved, and *efficiently*, such that performance requirements are met and the network lifetime is maximized. The need to manage a large collection of autonomous sensor nodes poses challenges from a programming perspective. State of the art programming languages and methods for sensor networks require the end user to manually translate the global application behavior in terms of local actions at each node, which is likely to be time-consuming and error prone for complex applications. Also, the application-level logic is tightly interfaced with the part of the program that coordinates lower level services such as resource management, routing, localization, etc. This lack of separation between system-level code and application-level code results in high complexity of coding non-trivial system behaviors.

There is a growing interest in *macroprogramming* of sensor networks [5, 10] which means moving beyond node-centric programming and instead specifying aggregate behaviors which are then automatically translated by a compilation framework into node-level specifications. This is motivated by the realization that the end user will be a domain expert and not a computer scientist, and will be primarily interested in the monitoring and control of physical phenomena. The details of in-network computing and communication which provides the desired functionality will be of incidental interest in most scenarios.

We introduce a macroprogramming model called the Abstract Task Graph (ATaG) that builds upon the core concepts of data driven computing and incorporates novel extensions for distributed sense-and-respond applications. The types of information processing functionalities in the system are modeled as a set of *abstract* tasks with well-defined input/output interfaces. User-provided code associated with each abstract task implements the actual processing in the system. An ATaG program is 'abstract' because the number and placement of tasks and the control and coordination mechanisms are determined at compile-time and/or run-time depending on the characteristics of the target deployment.

ATaG enables a methodology for architecture-independent development of networked sensing applications. Architecture independence is the ability to specify application behavior for a generic, parameterized network architecture. The same application may be auto-

*This work is supported in part by the National Science Foundation, USA under grant number IIS-0330445.

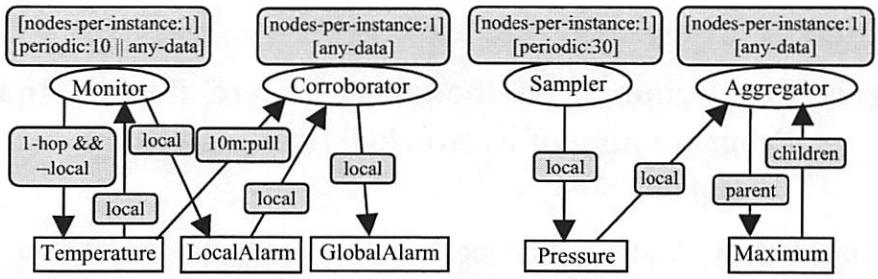


Figure 1: An ATaG program for environment monitoring

matically synthesized for different network deployments, or adapted as nodes fail or are added to the system. Furthermore, it allows development of the application to proceed prior to decisions being made about the final configuration of the nodes and network.

The focus of the ATaG approach is on simple specification of more complex patterns of information flow. A second objective is to define a process for automatically analyzing a user-supplied ATaG program and generating a deployment-specific distributed software system that consists of (a) the user-supplied application level code, and (b) a suitably customized runtime system responsible for control and coordination among the application level tasks. The present focus of our research is on defining the syntax and semantics of ATaG, and designing a runtime system and compiler for functionally correct translation of architecture-independent ATaG programs into architecture-specific node-level behaviors. Low level optimizations for a specific target platform have not yet been addressed.

Section 2 presents a sample ATaG program for an environment monitoring scenario with a view to highlight the key ideas before discussing them in more detail. Section 3 discusses the key concepts of ATaG and the syntax and semantics of ATaG programs. Details of the system level support for the ATaG model can be found in [1]. A brief overview of the end-to-end application development methodology is provided in Section 4. We discuss related work in Section 5 and conclude in Section 6.

2 An Illustrative Example

Figure 1 is an ATaG program for an environment monitoring system. The application is designed for a network of sensor nodes, each equipped with a temperature and a pressure sensor. The application exhibits two behaviors: the periodic computation and logging of the maximum pressure in the system, and the periodic monitoring of temperature. If the temperature gradient between a node and its neighbors exceeds a threshold, the node is required to corroborate the anomaly by surveying a larger area and then trigger an alarm. Corroboration helps to avoid false alarms due to a sensor malfunction.

The ATaG programmer first models each behavior in terms of a pattern of node-level interaction. In this case, the temperature monitoring requires a neighbor-to-neighbor exchange of temperature readings for gradient computation, and a many-to-one information flow for corroboration. The pressure monitoring and logging can be visualized in terms of information flow with incremental aggregation at each node of a virtual tree topology – a common pattern for efficient data aggregation.

The next step is to identify the types of processing and the types of data in the system, referred to in ATaG terminology as *abstract tasks* and *abstract data*. The abstract data for pressure averaging are: `Pressure` to represent the reading from the pressure sensor, and `Maximum` to represent the (partial) maximum. Similarly, the abstract data for temperature monitoring are: `Temperature` to represent readings from the temperature sensor, and `LocalAlarm` and `GlobalAlarm` to represent conditions where the threshold is violated and corroborated over a greater area, respectively. The abstract tasks for pressure monitoring are: `Sampler` to periodically record the pressure at the node, and `Aggregator` to track the maximum pressure readings from the node's children in the virtual tree. Similarly, the abstract tasks for temperature monitoring are: `Monitor` to compute local gradients, and `Corroborator` to analyze readings from the larger neighborhood. The programmer supplies the code for each abstract task and abstract data. This constitutes the imperative part of the ATaG program, and is also the *only code written by the programmer.*

The input/output interfaces of the abstract tasks are shown in the figure. Note that `Monitor` produces `Temperature` instances that represent local readings, and consumes `Temperature` instances produced by its neighbors. Since this distinction is between instances of data and not the type of data, the relationship between the abstract task and abstract data is both input and output.

The final step is to associate annotations (depicted by shaded, rounded rectangles) to indicate task placement and information flow patterns. For instance, the annotations indicate that `Monitor` is to be instantiated on

every node in the system, run periodically, and also executed when a new `Temperature` instance is available. The `Temperature` instance produced by `Monitor` is to transmitted to all 1-hop neighbors and not added to the local data pool. `Corroborator` will be triggered by the production of a `LocalAlarm` by `Monitor`, 'pull' all instances of `Temperature` within a 10 meter radius, and possibly produce a `GlobalAlarm`. For the other behavior, the `Aggregator` is to be executed whenever an instance of `Pressure` is produced locally (by the periodic `Sampler`) or an instance of `Maximum` is received from any of the node's children. The output of `Aggregator` is to be transmitted up the virtual tree maintained by the runtime.

3 Programming with ATaG

3.1 Key concepts

ATaG is based on two key concepts: data driven program flow and mixed imperative-declarative specification.

In *data driven computing*, tasks are passive objects that are defined in terms of their input and output interface. Tasks do not interact with each other. The basic primitives available to the programmer are `getData()` and `putData()` for consumption and production of data items respectively from the *data pool*. A task is automatically scheduled for execution when its operands become available. This scheduling is performed by an underlying runtime system that manages the data pool. The data driven paradigm is attractive for several reasons. Tasks can use data items at the desired level of abstraction without worrying about how they are produced. Programs are highly extensible and reusable because there is no direct task-to-task coupling. From an implementation perspective, data driven programming can be naturally supported by an event driven runtime system, resulting in efficient resource utilization. An 'event' is the production or consumption of a data item from the data pool.

Mixed imperative-declarative specification facilitates a clear separation of functionality from other non-functional aspects such as task placement and coordination. For sensor networks, this separation is especially critical because it allows the same program to be synthesized without modification onto various deployments by interpreting the declarative part differently. Also, we chose to design a visual programming interface for specifying the declarative aspect, thereby eliminating the need for programmers to learn a new syntax. Support for both network awareness and network transparency through such separation of concerns has been explored in the distributed computing community [6, 8] – our motivation is to design suitable mechanisms that are useful for networked sensing applications.

3.2 Syntax

ATaG captures global application behavior in a *network-aware* but *network-independent* way. The close coupling of sensor networks with the physical environment, and the dependence of in-network processing on the spatio-temporal location of data being processed mandate network awareness. On the other hand, requirements of portability and architecture independence require a model and representation that is network independent. Note that ATaG does not hide parallelism. The compiler translates a concise and architecture-independent but explicitly parallel specification into the node-level control and coordination behavior.

An ATaG program is a set of *abstract declarations*. An abstract declaration can be one of three types: *abstract task*, *abstract data*, or *abstract channel*. Hereafter, we occasionally omit the word 'abstract' for sake of brevity when the meaning is apparent. Each abstract declaration consists of a set of *annotations*. Each annotation is a 2-tuple where the first element is the *type* of annotation, and the second element is the *value*.

Abstract task: Each abstract task declaration represents a *type* of processing that could occur in the application. The number of instances of the abstract task existing in the system at a given time is determined in the context of a specific network description by the annotations associated with that declaration. Each task is labeled with a unique name by the programmer. Associated with each task declaration is an executable specification in a traditional programming language that is supported by the target platform. Table 1 describes the annotations that can be associated with a task declaration in the current version of ATaG.

Abstract data: Each abstract data declaration represents a *type* of application-specific data object that could be exchanged between abstract tasks. ATaG does not associate any semantics with the data declaration. The number of instances of a particular type of data object in the system at a given time is determined by the associated annotations in the context of a specific deployment and depends on the instantiation and firing rules of tasks producing or consuming the data objects.

Each data declaration is labeled with a unique *name*. Similar to the executable code associated with the task declaration, an application-specific *payload* is associated with the data declaration. This payload typically consists of a set of variables in the programming language supported by the target platform. No annotations are currently associated with abstract data items.

Abstract channel: The abstract channel associates a task declaration with a data declaration and represents not just which data objects are produced and/or consumed by a given task, but which instances of those types of data

Type: Instantiation	
value[:parameter]	**Description**
one-on-node-ID:*id*	Create one instance of the task on node *id*
one-anywhere	Create one instance of the task on any node in the network
nodes-per-instance:[/]*n*	Create one instance of the task for each *n* nodes of the network. When *n* is preceded by a "/", create exactly *n* instances of the task and divide the total number of nodes into *n* non-overlapping domains, each owned by one instance.
area-per-instance:[/]*area*	Same as for nodes-per-instance. Parameter denotes area of deployment instead of number of nodes. The non-overlapping domains are in terms of area of deployment, not number of nodes.
spatial-extent:$x_1, y_1, x_2, y_2, \ldots$	Create one instance of the task on every node that is deployed in the polygon defined by the co-ordinates (x_1, y_1), (x_2, y_2), ..., (x_1, y_1).

Type: Firing rule	
value[:parameter]	**Description**
periodic:*p*	Schedule task for periodic execution with period of *p* seconds.
any-data	Schedule task for execution when at least one of the input data items are available.
all-data	Schedule task for execution only when all the input data items are available.

Table 1: Abstract Task: Annotations

Type: Initiation	
value	**Description**
push	The runtime system at the site of production of each instance of the associated abstract data item is responsible for sending the instance to nodes hosting suitable instances of the consumer task(s).
pull	The runtime system at the node hosting an instance of the consumer task is responsible for requesting the required instance(s) of the associated abstract data item from the site(s) of production.

Type: Interest	
value[:parameter]	**Description**
[¬]local	Channel applies to the local data pool of the task instance. The negation qualifier excludes the local data pool, and can be used in conjunction with other qualifiers.
neighborhood-hops:*n*	Channel includes all nodes within the *n*-hop neighborhood of the node hosting the task instance
neighborhood-distance:*d*	Channel includes all nodes within a distance *d* of the node hosting the task instance
all-nodes	Channel includes all nodes in the system
domain	Channel includes all nodes that are owned by the task instance. This value is used in conjunction with the nodes-per-instance or area-per-instance values of the Instantiation annotation of the abstract task.
parent	Channel applies to the parent of the node hosting the task instance - in the virtual tree topology imposed on the network by the runtime system.
children	Channel applies to all children of the node hosting the task instance - in the virtual tree topology imposed on the network by the runtime system.

Table 2: Abstract Channel: Annotations

items are of interest to a particular instance of the task. Table 2 describes the annotations that can be associated with an abstract channel in the current version of ATaG. The abstract channel is the key to concise, flexible, and architecture-independent specification of common patterns of information flows in the network. For instance, spatial dissemination and collection patterns may be expressed using simple annotations such as "1-hop," "local," or "all nodes," on output and input channels. More sophisticated annotations may be defined as needed or desired for a particular application domain.

3.3 Semantics

The two basic primitives available to the programmer are `getData()` and `putData()` for consuming and producing data items. The runtime system manages the *data pool* and moves data between producers and consumers. We now briefly summarize the semantics of ATaG.

If the task is *periodic*, it is scheduled for execution when the periodic timer expires, regardless of the state of its input data items. The per-task timer is set to zero each time the task begins execution and is said to expire when the timer value becomes equal to the task's period. If the task is *any-data*, it is scheduled for execution as soon as a new instance of any of its input data items is available. If the task is *all-data*, it is scheduled for execution as soon as a new instance of each of its input data items is available. Other valid firing rules are *periodic* $\vee$ *any-data* and *periodic* $\vee$ *all-data*.

Each well-behaved task must invoke exactly one `getData()` call for each of its input data items, and may invoke at most one `putData()` for each of its output data items. `getData()` is a destructive read from the task's perspective. Once a particular instance of a data item is read by a task, it is considered to be eliminated from the data pool as far as that task is concerned.

Task execution is atomic. Each application-level task will run to completion before another application-level task can begin execution. All members of the set of dependent tasks of a particular data item are executed before other tasks that might be dependent on the output data items of the tasks in this set are executed. Whenever the production of an instance of a data item results in one or more of its dependent tasks to become ready, those tasks will consume the same instance when they invoke a `getData()` on the input data item. This means that the particular instance that triggered the task(s) will not be overwritten or removed from the data pool before every scheduled dependent task finishes execution. The implication of this is that `putData()` is not guaranteed to succeed if an instance of the abstract data item being produced cannot be overwritten. Such situations might arise if, for example, the rate of production of an abstract data item is greater than the rate of consumption. The application developer is responsible for checking for success or failure of `putData()`.

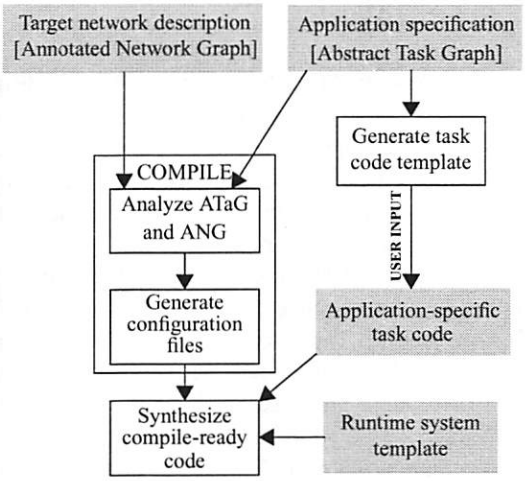


Figure 2: Application development with ATaG

4 Application Development Methodology

Figure 2 depicts the application development methodology using ATaG. The application developer graphically inputs the declarative part of the ATaG program and a description of the target deployment in the form of an annotated network graph (ANG), which is not discussed in this paper. The ANG contains information such as the number of nodes, the co-ordinates of each node, network connectivity, etc. A code generator analyzes the ATaG program, determines the I/O dependencies between tasks and data objects, and generates code templates for the abstract tasks and data. The programmer populates the code templates with application functionality. The compiler then interprets the program annotations in the context of the ANG, and generates configuration files for each node that customize the behavior of that node based on its role in the system. Finally, compile-ready code is generated for each node in the network.

The graphical interface to the programming and synthesis environment is through a configurable graphical tool suite called the Generic Modeling Environment (GME) [4]. The declarative part of the ATaG program which consists of the various declarations and their annotations is specified visually. In fact, we exploit the composability of ATaG and allow users to create libraries of ATaG programs that can be simply concatenated to build larger applications. GME stores the model defined by the user in a canonical format. Tools called *model interpreters* can read from and write to this model database. In our case, model interpreters were written for the components represented by unshaded boxes in Fig. 2.

5 Related Work

The core ideas of ATaG have been applied in different contexts in the distributed computing community. Modular, extensible, and convenient parallel programming was the motivation for the data driven paradigm in the Data Driven Graph [11] model. Separating the core application functionality from other concerns such as task placement and coordination was one of the primary motivations of efforts such as Distributed Oz [6] and IBM's PIMA project [2]. Tuple spaces [3, 9] provides a content addressable persistent shared memory for coordination across distributed processes. Although the notion of a shared data store also exists in ATaG, our 'active' data pool abstraction that schedules tasks based on the availability of data instances is fundamentally different from the 'passive' tuple space whose modifications are really a side effect of task execution on different nodes.

TinyDB [7], Regiment [10], Kairos [5], and Semantic Streams [12] are examples of macroprogramming methodologies for sensor networks. While ATaG explores a mixed imperative-declarative programming style and data-driven program flow, TinyDB provides a declarative SQL-like query interface for sensor data. Regiment is a demand-driven functional language based on Haskell, with support for region-based aggregation, filtering, and function mapping. Kairos is an imperative, control-driven programming paradigm that provides a distributed shared memory abstraction to the node level program. The Semantic Streams markup and declarative query language, based on Prolog, is used to specify queries over semantic information directly, while the actual selection, wiring, and optimization of low level modules to implement the querying is performed by a service composition framework.

6 Conclusion and Future Work

We have presented a novel method for specifying sensor network programs based on the Abstract Task Graph. ATaG programs consist of two parts: a declarative section, which specifies the connectivity between tasks and constraints on their placement and communication; and an imperative section, with the implementation of the tasks written in a traditional computer language. In the current system, these tasks are instantiated on the nodes at compile-time, but in future work, we plan to investigate fully dynamic versions, instantiating tasks based on the current network hardware and connectivity and the underlying measurements. This will probably be restricted to the more capable hardware platforms. In any case, the declarative specification is fully portable to both other network architectures and other hardware architectures, requiring at most a porting of the individual task code, which represents only a small fraction of the system functionality.

This paper focused on the key concepts of ATaG and the syntax and semantics of ATaG programs. We did not cover issues such as the operational semantics of the compiler and synthesis system, valid and invalid constructs in ATaG and opportunities for performance optimization in the runtime. Finally, the set of annotations and the synthesis and analysis tools available thus far are limited; for the platform to achieve wider applicability, we expect to extend these significantly in the future.

References

[1] A. Bakshi, A. Pathak, and V. K. Prasanna. System-level support for macroprogramming of networked sensing applications. In *Intl. Conf. on Pervasive Systems and Computing (PSC)*, June 2005.

[2] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: An application model for pervasive computing. In *6th Annual ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 2000.

[3] C. Curino, M. Giani, M. Giorgetta, A. Giusti, G. P. Picco, and A. L. Murphy. Tiny Lime: Bridging mobile and sensor networks through middleware. In *3rd IEEE Intl Conf on Pervasive Computing and Communications*, 2005.

[4] The Generic Modeling Environment, http://www.isis.vanderbilt.edu/projects/gme.

[5] R. Gummadi, O. Gnawali, and R. Govindan. Macroprogramming wireless sensor networks using Kairos. In *Intl. Conf. Distributed Computing in Sensor Systems (DCOSS)*, June 2005.

[6] S. Haridi, P. V. Roy, P. Brand, and C. Schulte. Programming languages for distributed applications. *New Generation Computing*, 16(3):223–261, 1998.

[7] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *2nd Intl. Workshop on Information Processing in Sensor Networks*, 2003.

[8] O. Holder, I. Ben-Shaul, and H. Gazit. Dynamic layout of distributed applications in FarGo. In *21st Intl. Conf. Software Engineering*, 1999.

[9] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications with the tota middleware. In *PerCom*, March 2004.

[10] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *1st Intl. Workshop on Data Management for Sensor Networks (DMSN)*, 2004.

[11] V. D. Tran, L. Hluchy, and G. T. Nguyen. Data driven graph: A parallel program model for scheduling. In *Proc. 12th Intl. Workshop on languages and Compilers for Parallel Computing*, pages 494–497, 1999.

[12] K. Whitehouse, F. Zhao, and J. Liu. Semantic streams: A framework for declarative queries and automatic data interpretation. Technical Report MSR-TR-2005-45, Microsoft Research, April 2005.

A Sensor-based, Web Service-enabled, Emergency Medical Response System

Nada Hashmi
10Blade, inc., nada@10blade.com
Dan Myung
10Blade, inc., dan@10blade.com
Mark Gaynor
Boston University, mgaynor@bu.edu
Steve Moulton
Boston University, smoulton@acs.bu.edu

Abstract:

Recent advances in low power wireless sensor networks are enabling new and exciting applications for wireless devices. At the same time, the power and flexibility of web services is expanding the power of the internet. Herein, we describe a scalable emergency medical response system that couples the efficient data collection of sensor networks with the flexibility and interoperability of a web services architecture

1. Introduction

Sensor networks consist of small, low-power, low-cost devices with limited computational and wireless communication capabilities. They represent the next step in wireless communication's miniaturization, and their power and size make it feasible to embed them into wearable vital sign monitors, location-tracking tags in buildings, and first responder uniform gear [1, 2].

Emergency Medical Services (EMS) systems need to communicate with hospitals from the field and exchange information about patient condition, expected time of patient arrival, and occasionally inquire about the ability to accept more patients [3]. An ideal EMS system should therefore provide real-time information and tracking of patients, staff and emergency vehicles. A web services architecture can meet these needs by addressing the problems associated with systematic application-to-application interactions over the web. This is because the key components of web services are a focus on interoperability and support for efficient application integration [4].

In this paper, we present a sensor-based, web service-enabled emergency medical response system. The paper begins with a brief overview of EMS systems in the United States. This is followed by a presentation of our system's architecture and data flow. Next, we review our preliminary experience with the system and lastly, discuss the future of this research.

2. Background

Emergency Medical Services (EMS) respond to sudden, unexpected events in oftentimes unfamiliar surroundings—where important information may be lacking or unknown until they arrive at the scene. The information gathered by Emergency Medical Technicians (EMTs) and paramedics is sometimes radioed ahead of the patient, depending on the severity of the patient's condition and the expected transport time. In all cases, the patient care report is summarized verbally during hand-off of the patient and a paper-based or electronic report later completed and added to the patient's hospital record. Within this process, tracking of the emergency response vehicle from the time of dispatch to scene arrival time, and finally to the health care facility is ad-hoc and poorly monitored [5].

In mass casualty situations, the current system often fails because EMTs and hospitals cannot effectively triage the injured and severely injured in a rapid, coordinated manner. Large numbers of victims can quickly overwhelm the triage process and prevent emergency field personnel and hospital staff from providing quality trauma care. Rapidly identifying and stratifying the most severely injured patients from those less severely injured poses a unique set of challenges, as does efficiently monitoring and transporting victims to an appropriate and awaiting trauma center.

The triage process has traditionally occurred at the hospital gate (typically at the entrance to the

Emergency Department). There, an emergency physician or experienced trauma surgeon stratifies patients with one or more triage tools based on mechanism of injury, physiologic criteria, injury site and severity, preexisting disease, age, and survival expectation. Over-triage, a common problem, can tie up valuable resources that could otherwise be made available for more severely injured patients. Emergency personnel must therefore decide, as early as possible, which patients will benefit most from transport to a dedicated trauma center and which patients require less immediate care [3, 5]. Clearly, there is a need for improved communication, documentation, and exchange of information between the pre-hospital and hospital phases of emergency care.

3. System Architecture

Figure 1 illustrates the overall system architecture of our emergency services application. The system has several major components including:

- A web services architecture to process, interpret, aggregate and present information
- A local command site for field coordination
- A central command site for global resource management
- Cellular/Satellite wireless links for real time communication between local and remote sites
- A wireless infrastructure for real-time data transport between motes and local PDAs and tablet PCs
- Patient sensors (a pulse oximetry sensor integrated with a GPS receiver, micro-processor, data storage & transmitter) for patient vital sign and location monitoring
- Wireless PDAs and tablet PCs for use by EMS field personnel

Data from the patient sensors is combined on the mote and forwarded to the ambulance base station via a proprietary protocol. The local EMTs and paramedics use PDAs or tablet PCs to enter patient information as patients are evaluated and treatment is provided in the field; this information is linked to a real-time sensor data timeline by the web service application. Each patient's medical history is available in the field (even when disconnected from the internet), and at the central server (if connected to the internet). All data exchange (except between the sensors and base station) is based on emerging web services standards, encouraging interoperability by supporting the straightforward integration of real-time sensor data into applications and the easy exchange of sensor data between applications. The overall system goal is to provide secure, end-to-end real-time (including medical, environmental, and geo-location) information to first responders, who can then provide situational awareness for local decision support and the global management of resources.

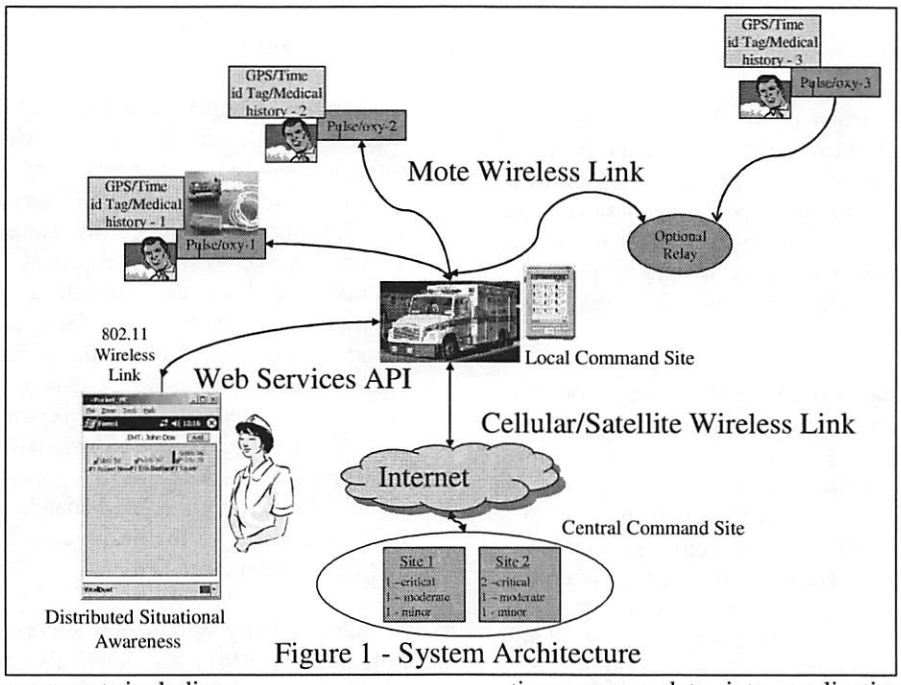


Figure 1 - System Architecture

Individual wireless, patient sensors are an important component of our system, for they provide real-time vital sign and patient location data both locally and

centrally. When a patient is moved to an ambulance or temporary command tent, the vehicle or command center GPS overrides the GPS on the patient, thereby providing location data for all patients at that site. At the same time, however, each patient's individual vital sign sensor continues to transmit unique, real-time, vital sign data.

The patient sensor, shown in Figure 2, is based on the MICAz mote, developed at UC Berkeley in collaboration with Intel Research. This device consists of an 8-bit Atmel ATMEGA 128L micro controller, 132K of memory, 512K of nonvolatile flash memory, and a 19.2 Kbps radio operating in the 2.6 GHz spectrum. These motes run the open source TinyOS operating system. The mote is interfaced to a pulse oximeter (BCI, Inc.) and the Crossbow MTS420CA sensor board, which has a Leadtek 9546 GPS module. In addition to a GPS receiver, the MTS420CA has an onboard dual-axis accelerometer, a barometric pressure sensor, humidity sensor, and temperature sensor. The mote transports its data to a laptop computer (interfaced to a mote) in the onsite ambulance via 802.15.4. These motes provide a powerful platform for experimentation with both digital and analog sensors.

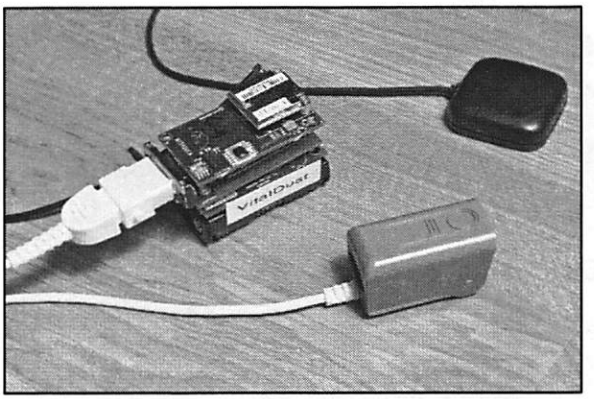


Figure 2: Patient mote with pulse oximeter and GPS sensors.

Sensor data must be "application friendly" to facilitate wide spread adoption of real-time sensor data within IT applications. Emerging standards for web service are a dominant design for the distributed exchange of data between applications. Our application has adopted these standards for both local and wide area exchange of real-time sensor data. Local personnel outfitted with applications on mobile devices have situational awareness with real-time access to sensor data via a web service. This local connectivity does not depend on a connection to the traditional internet. When connected via a cellular or satellite link, the real-time sensor data is available as a web service to centralized or distributed command centers. Compliance with emerging open standards (such as web services) enables a flexible architecture in the context of exchanging data between heterogeneous systems in both the local and wide area.

First responders provide situational awareness of the local, dynamic environment. Our system provides this with local PDAs and tablet PCs, which display data from local/remote sensors in real-time. These end devices have, in addition, embedded rules to help triage local patients based on their vital signs. Our end-user software provides several views of resources: a local view of patients being cared for a by single EMT, or a global view of all patients being treated by a group of EMTs. This allows a commander to "see" and respond to the needs of the EMTs and monitored patients, based on the available resources.

The central coordination of global resources is critical in the management of large scale (or multiple) events that span large geographical areas. In our system, aggregated sensor data is made available to a centralized, or group of distributed command and control centers, via web services. This allows for the development of command and control software in both Microsoft and Unix development environments.

Our emergency medical services application generates individual, electronic patient care records. These pre-hospital records are composed of manually entered patient information (history, physical exam findings, procedural information, and response to treatment) and automated sensor data. The manually entered patient information is captured in a SQL database and converted into an XML (HL-7) data format. The geo-location and pulse oximetry data (heart rate and blood oxygen saturation) generated by the motes is transmitted in XML and automatically integrated into each patient's electronic patient care record. These combined XML datasets can then be shared via web-service XML RPC calls. The resultant XML data is compatible and shareable with similarly structured web services. This means that the contextual healthcare data that is captured by our application could be combined with hospital medical record, laboratory, radiological, nursing and enterprise-wide demographic data. In the not too distant future, these large datasets will be mined for research purposes, provided the data is de-identified in accordance with current Health Insurance Privacy and Accountability Act (HIPAA) regulations.

4. Data Flow

Figure 3 illustrates the 3-layers of feedback and 5 categories of real-time dynamic data that drive the application's decision support system: at the edge, EMTs and paramedics have situational awareness with sensor data (1) and data from the local command center (2). The local command center receives patient specific healthcare and sensor data (3) from EMTs. At the highest layer the central command center receives aggregated data from each local site (5).

4.1 Edge Computing

The edge of an incident is where the EMTs and patients are physically located. Each EMT needs continuous access to real time sensor data from each patient to determine the triage order of all patients assigned to him or her. This data feed is represented as (1) in figure 3. It depicts the continuous stream of vital sign data from each patient under a specific EMT's control. There is also feedback from the EMT to the sensors (1). The sensor attached to each patient carries individual patient information. The primary concern in a crisis situation is that each EMT has situational awareness of their assigned patients based upon real time data (1). There is also a flow of data from the local command center to each EMT (2), which might be information about a noted change in a particular patient's vital signs, treatment suggestions, and/or instructions on where to transport the patient. Providing real-time data to the EMT allows for situational awareness of immediate time-critical responsibilities.

4.2 Local Command Center

Data flows from each sensor network (3) and is aggregated at the local command site along with data from each EMT (2). The local command center provides a view of all monitored patients in the sensor network, thereby allowing the effective management of local resources. One primary application at this layer utilizes a PDA to receive data from the local command center (4) to provide the EMT Commander a view of all local patients. Each local command center also receives data from the central command center for the effective coordination of global resources (5). The local command site enables a view of all local resources, along with data from the central command center.

4.3 Centralized

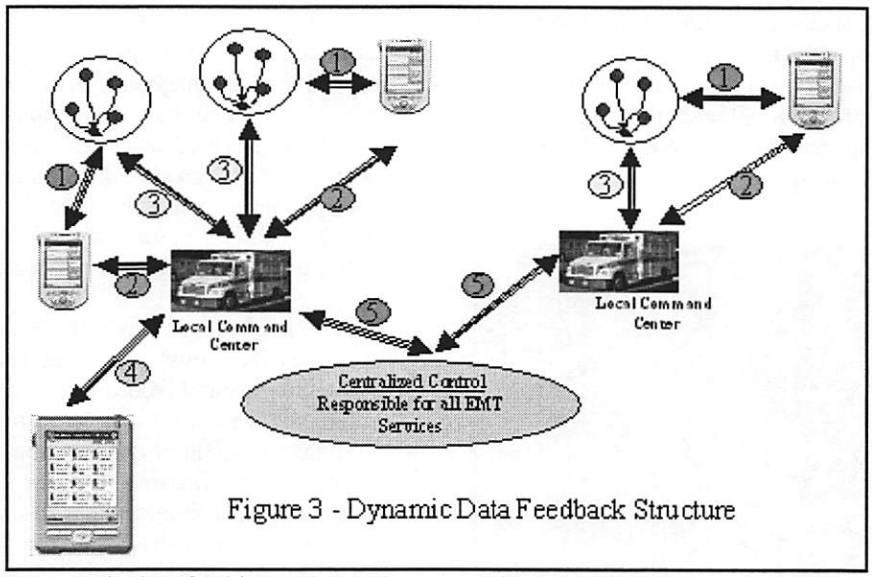


Figure 3 - Dynamic Data Feedback Structure

The top layer is the central command center. It acts as a web service client and receives data from each local site through the internet. Hence, there are no physical location limitations. This data is aggregated from each local command site (5) and includes real-time sensor as well as data input by local EMTs regarding the care of each patient. This data enables resources to be managed across a broad range of emergency events. The granularity of data required at the command center depends on the particular application. Systems that try to diagnosis or suggest treatment algorithms need fine grained data at the patient level. However applications for resource management across a distributed set of sites demands aggregated data in aggregated form. The overall architecture of our application enables data to flow between central and local command sites.

5. Evaluation

For proof of concept, an ambulance with two monitored "patients" was driven around our city. This urban setting, with its many high-rise buildings, was felt to be a challenging but realistic environment in which to test our wireless, sensor network infrastructure. We were especially interested in the accuracy and timeliness of the information relayed—both GPS location and vital sign sensor information. The two patients, each with a sensor mote, were loaded into the ambulance and GPS as well as vital sign (pulse oximetry and heart rate) data were relayed via the ambulance base station to a central command center. Time delays between the patient location (patient sensor) and actual location (ambulance GPS) were noted.

GPS satellites intentionally add a small error to civilian applications. While the accuracy is said to be about 100 meters, we were able to achieve accuracy to about 30 meters; the system used by the U.S. military has accuracy of 16 meters or less [6]. Pulse oximetry and heart rate were relayed in near real-time to the command center. There were, however, notable time delays of up to one full minute in relaying the GPS information. These time delays could be due to the implementation architecture of sensors or the use of web services over a wireless internet. Both Java and Microsoft .NET web service architectures were implemented.

While web services provide powerful and flexible service oriented architectures, they also introduce overheads such as the extraction of the SOAP envelope and parsing of the contained XML information. SOAP also requires typing information into every SOAP message, which creates bottle necks in synchronous applications. Binary data gets expanded (by an average of 5-fold) when included in XML, and also requires encoding/decoding of this data. These are the issues known over a wired internet. It is possible that these problems increase exponentially over a wireless internet, where there are bandwidth and connectivity issues. We are now in the process of conducting quantitative empirical studies to test web services over a wireless internet. The latency and through-put will be tested while the vehicle is standing still and at varying speeds. The data types and lengths will also be varied.

6. Future Work

Harvard University has an ongoing project called Hourglass, which is an internet-based infrastructure for connecting a wide range of sensors, services, and applications in a robust fashion [7]. In the coming year, we plan to integrate our system with Hourglass to take advantage of its scalable, robust architecture. Furthermore, the empirical experiments will provide guidance on optimizing our application to cater to various types of unexpected situations. Other sensors to collect other data such as EKG information will also be integrated. In addition, a recently formed partnership with our regional aero-medical group will serve as a real-life test bed for the evaluation of our system.

7. Conclusion:

Sensor networks combined with web services offers a unique system for EMS and other pre-hospital patient care systems. Our preliminary results suggest that the architecture described herein is sufficiently robust and scalable to meet the needs of our changing world.

References:

1. J. Hill et al., "System Architecture Directions for Networked Sensors," *Proc. 9th Int'l Conf .architectural Support for Programming Languages and Operating Systems* (ASPLOS 2000), ACM Press, 2000, pp. 93–104.
2. T.R.F. Fulford-Jones, G. Wei, and M. Welsh, "A Portable, Low-Power, Wireless Two-Lead EKG System," to be published in Proc. 26th IEEE EMBS Ann. Int'l Conf., 2004.
3. E.R. Frykberg and J.J. Tepas III, "Terrorist Bombings: Lessons Learned from Belfast to Beirut," Annals of Surgery, vol. 208, no. 5, 1988, pp. 569–576.
4. F. Curbera, W.A. Nagy, and S. Weerawarana. Web services: Why and how. In OOPSLA 2001 Workshop on Object-Oriented Web Services. ACM, 2001. pp. 34
5. Mann, Glenn Eric. Data Capture in the United States Healthcare System- the Pre-Hospital Phase of Patient Care. Masters Thesis. Boston University School of Medicine-Division of Graduate Medical Sciences. 2002
6. Highly, Sam. GPS use expanding by 2nd Lt. http://gislounge.com/freisin/gps.shtml
7. J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, M. Welsh. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications Harvard Technical Report TR-21-04

A Rule-based System for Sense-and-Respond Telematics Services

Jonathan Munson[†], SangWoo Lee[‡], DaeRyung Lee[‡],
David Wood[†], Gerry Thompson[†], Alan Cole[†]

[†]*IBM T. J. Watson Research Center, Hawthorne, New York*
[‡]*IBM Ubiquitous Computing Laboratory, Seoul*
jpmunson@us.ibm.com, lsw@kr.ibm.com, drlee@kr.ibm.com,
dawood@us.ibm.com, gerryt@us.ibm.com, colea@us.ibm.com

Abstract

We introduce a telematics-oriented event detection service, and programming framework supporting it, that enables application developers to more easily develop applications based on the sense-and-respond model. The system provides a rule-based programming model in which the application is partitioned in two parts: (1) a set of rules that operate on low-level position-update events and which, when triggered, produce high-level, application-defined events; and (2) logic that acts on the high-level events, which is deployed outside the event detection service, and typically within the environment of the enterprise deploying the rule. Programmers represent events of interest in the form of rules, which operate on both input received from the data acquisition systems as well as data resources provided and managed by the application programmers. Programmers declare the inputs the rules require, and the system is responsible for acquiring the inputs. A high-level programming framework assists programmers in defining the set of rules, and the actions that respond to events from the rules, and in deploying the rules to the system.

1 Introduction

In 2003 the Ministry of Information and Communication of the Republic of Korea formulated a strategy for the development of information technology, known as "8-3-9"—introducing and promoting *eight* services, building *three* infrastructures, and development of *nine* new growth engines. In 2004 the government of Korea and IBM jointly created the Ubiquitous Computing Laboratory in Seoul, where certain projects funded by the 8-3-9 program are being carried out. Telematics is both one of the eight services and one of the nine new growth engines, and is the subject of two of the UCL projects, which began in August 2004.

In this workshop we introduce one of those projects, the focus of which is to develop technology for supporting telematics-oriented sense-and-respond applications. We present an early result of the project, a prototype service we call the "Telematics Event Detection Service." The service enables application programmers to define situations of interest in the form of rules, and to deploy these rules to the Telematics Event Detection Service (TEDS). The service receives inputs from vehicles and from other data sources and evaluates the rules over them. When rules "trigger," notifications are sent to the applications that deployed them.

To test basic functions of the service, we created a simple fleet management application around the function known as "geofencing." The application expresses the geofences as TEDS rules, deploys them to TEDS, and in turn receives notifications when trucks enter and leave the geofence. The application is described in Section 3.1.

Geofencing is but the simplest of scenarios that we support with TEDS. We have identified a more complete set of target scenarios that TEDS should support, and then have used these to define the service's range of functionality.

1.1 Target Scenarios

IBM has many discussions with customers and potential customers about applications the customers are interested in. The following set of scenarios, which span a range of application domains, has been distilled from some these discussions.

Detecting changes in vehicle population density. A public-safety organization is interested in monitoring the deployment of its emergency vehicles to ensure that a vehicle can be dispatched to any location within a certain time. It would like to be alerted when the relative values between local densities change by a certain amount.

Track vehicle progress with respect to schedule. An operator of a fleet of delivery vehicles wishes to track its vehicles' progress with respect to their sched-

ules, and be alerted when a vehicle is more than a certain amount behind schedule. It may also wish to be alerted when a truck is returning empty to the warehouse so that it may begin preparing the truck's next load.

Location-based promotions. A marketing company handling promotions for certain establishments would like to be able to notify consumers as they approach those establishments with special promotions in effect at that time. It wants to send the promotions only to those who have accepted the service and who have been receptive to such promotions in the past, and it needs to avoid sending the promotion repeatedly.

Location-based warnings. A highways department in a fog-prone area wishes to warn motorists of dangerous fog conditions ahead so that the motorists will slow down. They need to be able to create such warnings quickly, and they need be able to distinguish between vehicles moving toward the conditions and those moving away.

Congestion detection. A highways department would like to have real-time congestion maps of area highways, and to be alerted when aggregate speeds in particular areas fall below a certain threshold.

Monitoring speed of fleet vehicles. An operator of fleet of delivery vehicles would like to ensure that its drivers obey posted speed limits Enterprise user defines the set of vehicles to monitor, and creates rule that will periodically compare their speeds with the speed limits in effect at the location where the speed was recorded.

Detection of excessive speeding. Public safety officials wish to know when aggregate speeds in a region are excessively high, either because of the accident rate in the area or because of particular road conditions.

Proximity to other mobile entity(ies). A rule is defined that when a certain number of the friends are within a given distance, causes a message to be sent to the user.

Pervasive gaming. An automobile club wishes to sponsor a combination road rally/treasure hunt, where members pair up and follow successive clues to a prize, but are penalized for exceeding speed limits.

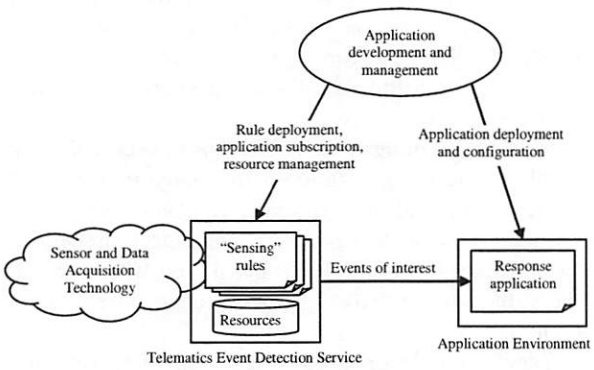


Figure 1. TEDS logical programming model.

1.2 The Value of a Shared Infrastructure

We believe there may be substantial value in a general-purpose, shared, infrastructure that could support any and all of the above services, simultaneously, over a large set of vehicles. Such an infrastructure would enable service providers to reach a broad customer base, without requiring an investment in their own infrastructure. With many services using the infrastructure, no one service must bear the entire load.

2 The Telematics Event Detection Service

Reflecting the structure of sense/respond applications, TEDS offers a rule-based programming model in which the application is partitioned in two parts: (1) a set of rules that operate on low-level position-update events and which, when triggered, produce high-level, application-defined events; and (2) logic that acts on the high-level events, which is deployed outside TEDS, and typically within the environment of the enterprise deploying the rule. Figure 1 illustrates.

As shown in Figure 1, programmers represent events of interest in the form of rules, which "trigger" (return true) when the events are detected. Rules operate on both input received from the data acquisition systems as well as data resources provided and managed by the application programmers. An example of an application resource is the geographical polygon describing a warehouse site's area, which would be used in a rule detecting a truck's entrance to the site. Rules and rule resources are described in Sections 2.1 and 2.5.

Applications subscribe to a rule to receive notifications of its triggering. The notification received includes any results that may have been computed in the evaluation of the rule. For example, a rule that determines if a subscriber's position lies within any zones that have a promotion associated with them will, if the position is within any such zones, include the set of zones (via identifiers) in the notification to subscribed applications. Applications may optionally choose to be notified only when the triggering status with respect to a particular subscriber changes, either from false to true, or true to false. For example, an application subscribing to a rule that detects entry to a warehouse site may wish to receive notifications only upon a truck's entry and exit of the site.

2.1 Rules

TEDS rules are essentially condition/action specifications, and may be as simple as a single Boolean expres-

sion, or a more complicated program with internal state. A rule condition is a logical expression (i.e., evaluates to true or false) composed of spatial and temporal logical functions joined through AND, OR, and NOT operators, and scalar functions joined with the usual relational operators. TEDS offers a number of built-in functions, which are described below.

Rule inputs include an entity's location (along with speed, heading, and accuracy estimates) received from the positioning technology, and any other input data available from the data acquisition technology.

2.2 Rule Functions

The spatial rule functions offered by TEDS operate implicitly on a subscriber position report, and have parameters associated with them that will also be input to the function. The *polygonID* and *pointID* parameters are identifiers for polygons and points. Polygons and points are specific kinds of rule resources, support for which is discussed in Section 2.5. **Table 1** below defines built-in logical functions of TEDS; **Table 2** defines built-in scalar functions.

Table 1. Built-in Logical Spatial Functions.

Name	*Parameters*
Description	
containedIn	polygonID | polygonSetID
True iff the entity position is contained in one or more of the given polygon or set of polygons.	
within DistanceOf	(pointID | pointSetID) distance
True iff the entity position is within the given distance from the given point or set of points.	
within DistanceOf	(entityID | entitySetID) distance
True iff the entity position is within the given distance from the position of the given entity or one or more of the given set of entities.	
hasIdentity	entity | entitySet
True iff the entity has the given identity or is a member of the one or more of the given entity sets.	
<, <=, ==, !=, >=, >	scalar_expr1 scalar_expr2
True iff the given relation between the two scalar expressions is true.	
elapsed	timer_identifier
True iff the timer identified has expired since the last time the expression was evaluated. Timers are declared, set, and reset explicitly, outside of condition expressions, in a language-specific manner.	
before/after	time
True iff the timestamp of the SPR is before/after the given time.	

Table 2. Built-in Scalar Functions.

Name	*Parameters*
Description	
distance From	point | susbcriber
Returns distance (in meters) from the given entity position to the specified point or entity.	
distance Traveled	point (route | path)
Returns the distance the entity has traveled from the given point along the given route or path. A route object is a precomputed route the susbscriber is expected to take; a path is a series of positions recorded for the entity. Use of a path object generates an implicit rule that will record the entity's positions.	
distance Traveled	time (route | path)
Returns the distance the entity has traveled since the given time along the given route or path.	

2.3 Rule Language

In the course of developing TEDS, we used multiple rule languages. Our first "language" was the construction of rules through building Java object structures of rule objects, much like an expression tree. We considered this to be not sufficiently easy to use. We then developed our own rule language and compiler. In the end, however, we realized that our primary value lay not in the rule language itself, but rather with the functions we provide outside the rule engine, such as rule optimization, application resource management, trigger reporting, and subscriber management (all discussed in later sections). Thus we adopted the approach of embedding existing rule engines within TEDS. This not only enables us to leverage the strengths of the particular rule engine in use, but gives us more flexibility in deployment as well.

The rule language we currently use is ABLE [3], an environment for building intelligent agents. ABLE supports different kinds of rule programming, through the variety of inference engines it provides. These include backward chaining, forward chaining, Rete networks, and simple sequential evaluation, among others. Below is an ABLE rule set that triggers when any one a set of trucks is in a no-standing zone.

```
ruleset FMGeofence {
  import
com.ibm.locutil.able.SubscriberPositionUpdate;
  import com.ibm.locutil.able.LUContext;
  import com.ibm.locutil.able.LURuleResults;
  import java.util.ArrayList;
  variables {
    SubscriberPositionUpdate sub;
    LUContext luCtx;
    LURuleResults results = new LURuleRe-
sults();
    ArrayList fcnResults = new ArrayList();
  }
  inputs { sub, luCtx }
  outputs { results }
  void process() using Script {
  : if (sub.isMemberOf("DeliveryShift1") &&
        sub.containedInPolygonSet(
           "FM:NoStandZones", fcnResults))
    then {
      results.setDidTrigger(true);
```

```
            results.addResult("zones", fcnResults);
        }
    }
}
```

By comparison, the same rule with our own (now abandoned) rule language is:

```
memberOf("DeliveryShift1")
&& containedInPolygonSet(("FM:NoStandZones")
```

Despite the obvious loss in succinctness by embedding a "foreign" rule engine such as ABLE, we still feel the advantages in flexibility outweigh the disadvantages. And for more complex rule sets, such as those that maintain local state, the disadvantages become less apparent.

We will continue to evaluate other rule languages and frameworks for use with TEDS, some of which are described in the section "Related Work." We are interested in rule frameworks that provide complementary value, such as strong support for temporal patterns, and that enable programmers to produce readable, easy-to-understand rule sets.

2.4 Subscriptions

Subscriptions to rules exist independently of the rules themselves. Subscriptions contain a number of parameters, including the rule being subscribed to, the address to send rule-triggering events to, and the report type, which is *full*, or *delta*. Full reports consist of the current set of subscriber position reports for which a rule was satisfied; delta reports consist only of the subscriber position reports that resulted in a change in the result of the rule evaluation with respect to a particular subscriber. Other parameters are not described here.

2.5 Rule Resources

Rules typically involve functions that relate data in the subscriber position report to other data. This data may be provided by the application, such as geometries to compare a subscriber's location to, or a set speed threshold to compare the subscriber's speed to. Or, the data may be generated by repeated evaluation of the rule. TEDS provides two forms of data store for these purposes.

Static Data. TEDS offers a simple data store to provide persistence for the data referred to in rule parameters. The global store is accessed implicitly in the *polygonSet* and *pointSet* parameters, and may be accessed explicitly as well (functions not shown). The data store may also be accessed from outside rules, through the TEDS application interface. The data store is not typed, although geometry types for points and polygons are stored differently in order to take advantage of databases with spatial functions and the spatial indexing supporting them.

Mobile Data. TEDS also provides a form of persistent data that is associated with particular subscribers. For example, a rule may wish to record a subscriber's time of entrance to a particular geographical area so that it can determine the total time a subscriber has been in the area. For purposes such as this, TEDS provides a data store associated with each subscriber.

3 TEDS First Prototype

Figure 2 illustrates the architecture of our prototype Telematics Event Detection Service. The main component is the Telematics Event Detection Engine (TEDE). Through a TEDE client, an application submits rules and provisions any resources required by the rule. Rule triggering events are received from the TEDE asynchronously through an interface the application must provide.

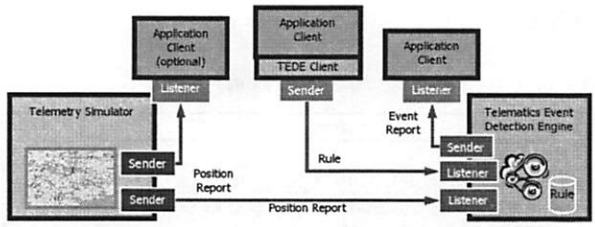


Figure 2. TEDS Demonstration Prototype.

As we as yet have no actual set of vehicles to receive input from, we developed a telemetry simulator that enables vehicle movements to be simulated. Through a map-based interface the simulator operator creates paths on the map, giving each segment its own speed.

3.1 Example: Fleet Manager Application

We developed "Fleet Manager" as an example of an application that takes advantage of the high-level event detection capabilities of TEDS. It supports various fleet management scenarios where fleet operators wish to be notified when vehicles of their fleet enter or leave certain geographical areas. It enables operators to quickly and easily define the areas and the vehicles to be monitored. It then transforms this information into a rule before deploying to TEDS. One of our objectives in developing this application is to explore high-level rule creation tools.

The Fleet Manager Application display (Figure 3) shows real-time positions of vehicles on the map and also represents locations symbolically by showing as-

sociations between vehicles and zones, as shown in the lower right window of Figure 3.

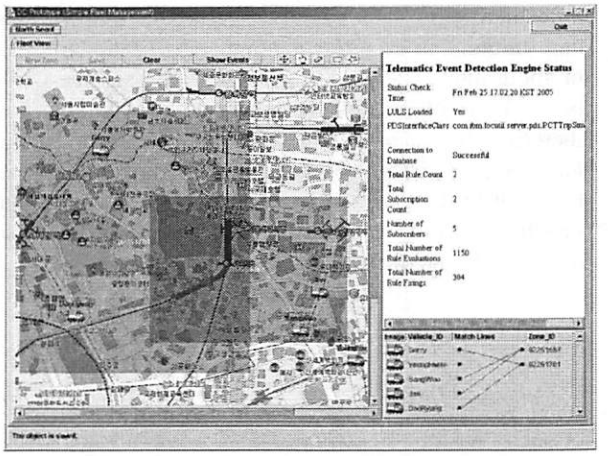


Figure 3. Fleet Manager application.

3.2 Lessons from Prototype

In developing the Fleet Manager application for the first TEDS prototype, we realized it had a number of shortcomings. First is that application programming requires a high level of effort to create interfaces to the TEDS system. TEDS provided no support for inserting rules into the system or for receiving events from the system, thereby requiring low-level programming from the application developer. Second, TEDS provided no framework to support the sense-respond programming model. Third, the prototype did not sufficiently support certain important features, such as subscriber groups.

These shortcomings are being addressed in the second prototype. In addition, we are also addressing the fundamental way input data is acquired, we are enabling rules to operate on a wider range of inputs, we are enabling the system to acquire input data from a wider range of sources, and we are also supporting application-provided rule functions. This new functionality is briefly discussed in the following section.

4 TEDS v2 and the S&R Framework

To make it easier to develop applications based on the sense/respond model, we have developed a framework providing both develop-time and run-time support. The framework, called the S&R Framework, relates to TEDS as shown in Figure 4 below.

4.1 S&R Framework

The S&R Framework is analogous to the Struts framework for developing Web applications based on the Model-View-Controller paradigm. With the S&R Editor, the developer defines the set of rules, and the actions for responding to events from the rules, that constitute a sense/respond application. The SNR file produced by the editor, together with the rule files and respond-action classes referenced by it, are then deployed to the Subscription Set Manager, which interfaces to TEDS to deploy rules and subscribe to them, and to the S&R Respond Controller, which receives the rule-triggered events from TEDS and invokes the related respond classes.

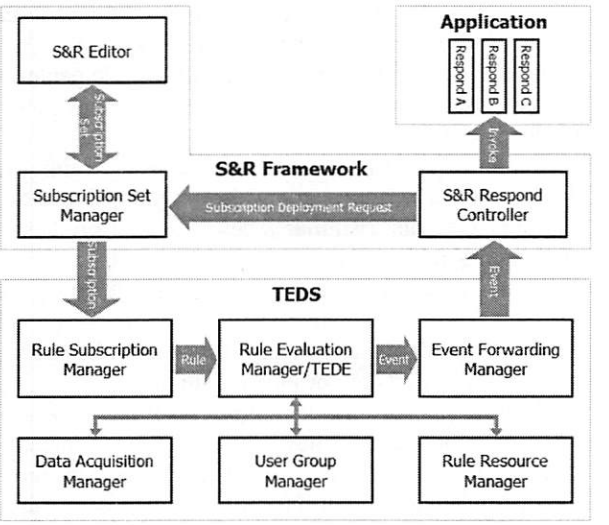


Figure 4. TEDS and S&R Framework

4.2 Extended Rule Inputs

The new TEDS architecture enables rule programmers to define rules that operate over a range of inputs, beyond the location input supported by the first prototype. Rule inputs may include any sensor data received from mobile entities such as fleet vehicles, as well as business-related data such as "pickup needed" signals.

Acquisition of these inputs is the responsibility of the Data Acquisition Manager, shown in Figure 4, Given the list of inputs a rule requires, and the rule's specified evaluation frequency, the DAM is responsible for acquiring the inputs at the rate specified and supplying them to the Rule Evaluation Manager.

5 Related Work

As a framework for programming sense-and-respond applications, TEDS is related to a broad array of work in real-time monitoring, event-driven systems, context-aware computing, and active databases. However, TEDS has a focus on those situations involved in telematics. Furthermore, it does not address applications where extended and complex patterns of events are of interest. Space does not permit a full discussion

of related technologies, but we briefly note those that are most closely related.

Chandy et al [3] describe an abstract programming model for dynamic applications that corresponds closely with the programming model of TEDS. The iSpheres Halo [9] platform offers a complete system for sense-and-respond programming. It does not, however, offer specific support for spatial events.

Some work has addressed the specific application of location-based notification. Chen et al [5] describe a publish/subscribe system for spatial triggering, focusing on efficient matching algorithms. Munson and Gupta [10] describe another spatial triggering system, focusing on an architecture for scalable implementation of this system over millions of users. This work is a predecessor of the system described here.

A set of spatial predicates and means of composing them are offered by Bauer and Rothermel [1] and Nelson [11] presents a similar a set of spatial predicates, and extends this with some temporal operators.

Stronger support for event patterns is offered by iQL [6]. iQL's functions and operators apply to generic numeric and textual data, and it does not offer specific support for spatial data or location input.

Houdini [8] is a rule language and framework used for user-preferences policy management for telecommunication services. As such, it addresses a different set of applications than TEDS, and does not address spatial events.

Amit [1] is a rule-based framework for detecting situations over potentially long-running event streams. Like Houdini, it could also serve as a base rule engine for TEDS.

6 Conclusions and Future Work

We have described the Telematics Event Detection Service, an infrastructure that we believe can enable a wide range of event-driven telematics services. The rule-based programming model of TEDS is a key feature that we plan to exploit further to address scalability. We believe it will enable the units of computation to be both replicated widely to make efficient use of resources, and pushed out close to the sensors to reduce communication needs in the wide-area network.

We have also described our initial effort at developing a high-level programming framework around the sense-and-respond application model. We believe that frameworks such as this are an important element in enabling programmers to exploit this model. We hope to extend the S&R Framework with high-level support for the development of rules themselves.

A primary goal for the future is to address the issue of scalability. We believe the flexibility of our service may make it attractive for service providers with large subscriber bases, such as large telematics service providers or wireless carriers. It would enable them to offer a wide range of services to their subscribers. For that to be feasible, however, our service must be scalable to a subscriber base numbering in the millions, and the number of services numbering in the hundreds or thousands. The resulting load on the service could be millions of rule evaluations per second. We must therefore have an architecture that can scale to this load, not only in the computational load of rule evaluation, but in the data transmission load involved in getting inputs to the rule engines. We plan to address this in a third phase of the project.

7 References

1. Adi, A., Etzion, O. Amit – The Situation Manager. The VLDB Journal, Springer-Verlag, Heidelberg, Vol. 13, No. 2.
2. Bauer, M., Rothermel, K. Towards the Observation of Spatial Events in Distributed Location-Aware Systems. In Proceedings of the 22nd International Conference on Distributed Computting Systems Workshops, 2002.
3. Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills, W. N. III, Diao, Y. ABLE: A Toolkit for Building Multiagent Autonomic Systems. IBM Systems Journal, Vol. 41, No. 3, 2002.
4. Chandy, K.M., Aydemir, B.E., Karpilovsky, E.M., Zimmerman, D.M. Event-Driven Architectures for Distributed Crisis Management. Presented at the 15th IASTED International Conference on Parallel and Distributed Computing and Systems, November 2003.
5. Chen, X.Y., Chen, Y., Rao, F., An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services. Proceedings of Second International Workshop on Distributed Event-Based Systems, San Diego, 2003.
6. Cohen, N.H., Lei, H., Castro, P., Davis, J.S. III, Purakayastha, A. Composing Pervasive Data Using iQL. In Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, June 2002, Callicoon, NY.
7. Federal Communications Commission. Enhanced 911. http://www.fcc.gov/911/enhanced/
8. Hull, R., Kumar, B., Lieuwen, D., Patel-Schneider, P.F., Sahuguet, A., Varadarajan, S., Vyas, A. "Everything Personal, Not Just Business": Improving User Experience Through Rule-based Service Customization. In International Conference on Service Oriented Computing (ICSOC 2003), Rome, December, 2003.
9. iSpheres Corporation. Halo™. http://www/ispheres.com.
10. Munson, J.P., Gupta, V.K. Location-Based Notification as a General-Purpose. In Proceedings of the Workshop on Mobile Commerce, Atlanta, 2002.
11. Nelson, G.J., Context-Aware and Location Systems. Ph.D. Thesis, University of Cambridge Computer Laboratory, 1998.

Meteorological Command and Control:

An End-to-end Architecture for a Hazardous Weather Detection Sensor Network

Michael Zink, David Westbrook, Sherief Abdallah, Bryan Horling, Vijay Lakamraju, Eric Lyons, Victoria Manfredi, Jim Kurose
Dept. of Computer Science, and
Center for Collaborative Adaptive Sensing of the Atmosphere
University of Massachusetts Amherst, MA 01003

Kurt Hondl
National Severe Storms Laboratory
National Oceanic and Atmospheric Administration
Norman, OK 73019

Abstract— We overview the software architecture for a network of low-powered radars (sensors) that collaboratively and adaptively sense the lowest few kilometers of the earth's atmosphere. We focus on the system's main control loop – ingesting data from remote radars, identifying meteorological features in this data, and determining each radar's future scan strategy based on detected features and end-user requirements. Our initial benchmarks show that that these components generally have sub-second execution times, making them well-suited for our NetRad system.

I. Introduction

Distributed Adaptive Collaborative Sensing (DCAS) of the atmosphere is a new paradigm for detecting and predicting hazardous weather using a dense network of low-powered radars to sense the lowest few kilometers of the earth's atmosphere [McLaughlin 2005]. *Distributed* refers to the use of large numbers of small radars, spaced close enough to "see" close to the ground in spite of the Earth's curvature and avoid resolution degradation caused by radar beam spreading. *Collaborative* operation refers to the coordination (when advantageous) of the beams from multiple radars to view the same region in space, thus achieving greater sensitivity, precision, and resolution than possible with a single radar. *Adaptive* refers to the ability of these radars and their associated computing and communications infrastructure to dynamically reconfigure in response to changing weather conditions and end-user needs. The principal components of the DCAS system include sensors (radars); meteorological algorithms that detect, track, and predict hazards; interfaces that enable end-users to access the system; storage; and an underlying substrate of distributed computation that dynamically processes sensed data and manages system resources. NetRad is a prototype DCAS system whose goal is to detect a tornado within 60 seconds of formation and to track its centroid with a temporal error no greater than 60 seconds. At the heart (or perhaps more appropriately, the "brains") of NetRad is its Meteorological Command and Control (MC&C) component that performs the system's main control loop – ingesting data from remote radars, identifying meteorological features in this data, reporting features to end-users, and determining each radar's future scan strategy based on detected features and end-user requirements. In this sense, NetRad is truly an end-end system, from the sensing radars through the computing and communication infrastructure and algorithms, to the end users. In this paper, we describe the software architecture of NetRad's MC&C and present initial benchmarks of its computational/communication requirements and performance. The important components of the MC&C that we study in the paper are *(i)* the data ingest, field retrieval, and meteorological detection algorithms, *(ii)* a feature repository that maintains a multi-level grid of feature values with associated user-based utilities, and that generates new sensing tasks for the networked radars, and *(iii)* a resource allocation/optimization process that determines the radars' scan strategy for the next system heartbeat. We discuss event notification mechanisms, and the computation of user-based utilities for competing sensing requests. We also discuss how NetRad timing considerations are addressed by structuring the feature repository as a blackboard system that temporally decouples data ingest/processing from the generation/optimization of future sensing activity. Our initial benchmarks, obtained by evaluating NetRad components using reflectivity and wind velocity data from the NOAA NEXRAD WSR88D system [NOAA 2005] show that that these components generally have sub-second execution times, making them well suited for use in the NetRad system.

II. NetRad System Overview and the MC&C Architecture.

We are currently building a NetRad prototype system to be deployed in southwestern Oklahoma, consisting of four mechanically scanned X-band radars atop small towers, and a central control site (later to be decentralized as the number of radars increases) known as the System Operations and Control Center (SOCC). The SOCC

consists of a cluster of commodity processors and storage on which the MC&C components execute.

NetRad radars are spaced approximately 30 km apart from each other and together scan an area of 80km x 80km and up to3 km in height. Each radar is tasked to scan an angular sector of up to 360 degrees in 1-degree increments, with a range gate (radial voxel) size of 100 meters out to 30 km. With two elevation scans during each tasking, each radar can thus produce up to 360*300*2 = 216K reflectivity and velocity values each time it is tasked. While existing meteorological radar systems such as NEXRAD generally operate in "sit and spin" mode (taking full 360-degree volume scans independently of location and type of meteorological features present), NetRad radars are tasked by the MC&C to focus on volumes of high interest to end-users, as discussed below.

Each radar consists of three subsystems:

- The **Rotating Tower Top** houses the radar unit and an embedded system that monitors the radar's operational parameters and enables operator actions in the case of anomalies (e.g., mechanical problems).
- The **Non-rotating Tower Top Subsystem** is located below the rotating joint and houses a data acquisition board (based on Field Programmable Gate Array (FPGA) technology), a radar controller, and a Gigabit Ethernet switch. The FPGA processes raw digitized data (at a rate of approximately 100 Mbps) into packets that are sent via the switch over a fiber optic cable to the Tower Base Subsystem. The radar controller controls the movement of the radar pedestal.
- The **Tower Base Subsystem** consists of a compute cluster (currently just a single node) and an IDE RAID storage system, connected via a Gigabit Ethernet switch to a router. The tower base takes raw radar data, computes so-called moment data (essentially an average of multiple radar-pulse measurements for a given voxel of space), while performing quality control (e.g., attenuation correction and range folding) on this data. The 1 Mbps moment data is sent to the SOCC over OneNet [OneNet 2005], an IP network operated by the Oklahoma state regents, which is configured to provide 4 Mbps connectivity from each radar to the SOCC. Raw data is archived at the tower base storage and can be transferred to SOCC storage in the background.

The SOCC is a centralized compute cluster (later to be decentralized) interconnected via a Gigabit switch, on which the MC&C algorithms execute. As shown in Figure 1, the SOCC has five main components *(i)* data ingest and storage, *(ii)* meteorological feature detection and multi-radar merging, *(iii)* feature repository, *(iv)* utility and task generation, and *(v)* optimization. We describe each of these functions below in more detail, roughly following the path taken by radar data through the MC&C, and the resulting re-tasking of the radars.

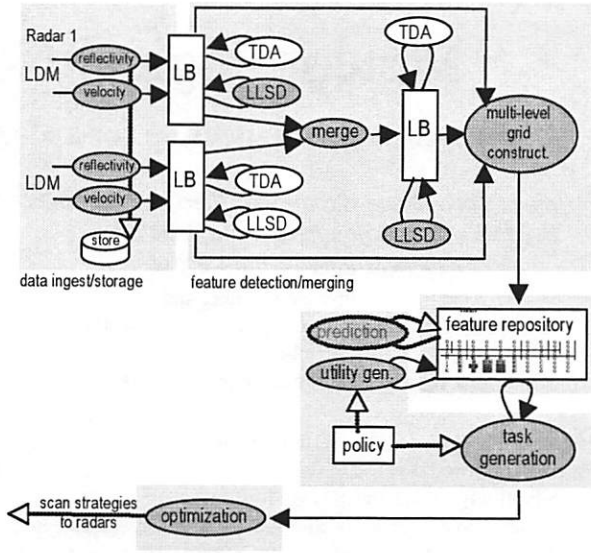


Figure 1: MC&C details and measurement points

A. *Data Ingest and Storage*

One SOCC computer is responsible for data ingest, archiving and distribution. Here, moment data (as well as the higher rate raw data, which is transferred at low priority) is streamed from the sensor nodes to the MC&C detection algorithms, and written to storage. In the future, both moment and raw data will be available to end-users via a query interface. Data from each elevation scan is sent from a remote radar to the MC&C data ingest routines using LDM [LDM 2005], client/server middleware that reliably transfers radar data over a TCP connection. Given the pre-provisioning of OneNet bandwidth for NetRad use, congestion loss is not a concern in our initial testbed. However, we are currently developing a UDP-based transport protocol that uses application-specific selective dropping for congestion-control in bandwidth-constrained environments [Banka 2005]. As illustrated in Figure 1, the per-radar received reflectivity and wind velocity data for an elevation scan are converted to NetCDF format, and stored in a file. An event is then posted and distributed among the MC&C procedures using the Linear Buffer (LB) pub/sub construct of the WDSSII software [Hondl 2003]. In section 3, we report the time needed to transfer an elevation scan from sender to receiver over a Gigabit LAN switch using LDM, write the associated files and post/distribute an event.

B. *Meteorological Feature Detection, Multi-radar Data Merging*

Once the notifications of available per-elevation reflectivity and wind velocity data have been posted,

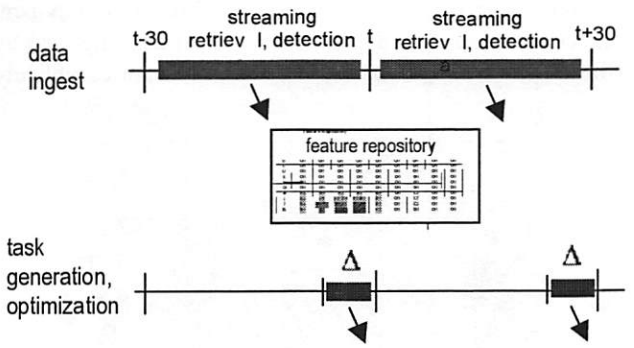


Figure 2: Feature repository: decoupling data ingest processing from periodic generation of radar commands

various meteorological detection algorithms can then read this data and perform feature detection. Figure 1 shows two such algorithms: *LLSD* (for which we provide per-elevation execution times in Section 3) computes wind shear and rotational divergence; *TDA* performs tornado vortex detection. Other per-radar detection algorithms can be easily "plugged in" using the LB pub/sub mechanism. Once a detection algorithm completes its execution, notification is provided through the LB. The *merge* procedure converts polar coordinate data to latitude/longitude coordinates, and fuses together spatially overlapping data from multiple radars. We benchmark the performance of the *merge* routine in Section 3.

C. *The Feature Repository*

NetRad system is a "real-time" system in the sense that radars must be re-tasked by the MC&C every 30 seconds – the system "heartbeat" interval. This heartbeat interval was chosen based on the physical properties of the mechanically-scanned radars, the timescale over which atmospheric conditions change, and the system goal of detecting and tracking tornados within 60 seconds. A notion of heartbeat also allows the radars to easily synchronize their operation (e.g., having overlapping radars scan the same volume in order to perform 3D wind retrieval), and also helps simplify the optimization of radar targeting. As we evolve from mechanically-scanned radars to rapidly reconfigurable solid-sate radars, we expect to relax the notion of a system heartbeat. As discussed above, radars are retasked based on detected meteorological features and the projected future evolution of these features. In order to decrease the timing dependencies between the ingest/processing of radar data and the generation of radar commands, the MC&C adopts a blackboard-like architecture [Jaganathan 1989]. At the heart of the MC&C is the feature-repository, a multi-level grid that stores both the underlying per-voxel reflectivity and wind velocity data, as well as higher-level spatially-coherent meteorological "objects" such as storms cells, areas of high wind shear or precipitation, and tornados. Each object also has a position, a spatial extent for non-point objects, and a tag representing the meteorological phenomenon that the object represents (e.g., storm-cell, mesocyclone, and tornado). The multi-level grid-construction procedure writes this information into the feature repository as needed data becomes available via the linear buffer, as shown in Figure 2. The generation of radar commands (the lower half of the control loop in Figures 1) proceeds *asynchronously* from the input processing of data (the upper half of the control loop). In this decoupled architecture, detection algorithms continuously post their results to the feature repository. As shown in Figure 2, at 30 second intervals the task generation component posts a set of tasks based on current state of the feature repository, and the optimization component then processes this task set and generates a scan strategy for the radars for the next 30 second cycle. In this design, we have relieved the time pressure on the detection components and somewhat relieved the time pressure on the MC&C components, task generation and optimization. One consequence of this design is that data that is not processed and posted on the feature repository before the task generation begins will not be acted upon until the next cycle of the system. This allows the system to avoid stalling, while waiting for late-arriving data (e.g., due to unanticipated network and processing delays). We are interested in the effects of this "decision lag" and also its relationship to the selection of the value of the system heartbeat. Figure 2 illustrates that the data-driven streaming retrieval and detection algorithms write the results of their execution into the feature repository. Starting Δ time units (where Δ is the execution time of the task generation and optimization algorithms) before the radars are to be re-targeted, the task generation process executes, followed by the optimization process. These processes may use all data available (in both the current time step, and previous time steps) in their computations. Once the optimization process has completed, the radars are then re-targeted for the next 30 second cycle. We note here that the feature repository is the central system "data structure." It is from here that meteorological objects can be obtained and subsequently delivered/displayed to end users. It is here that assimilated exogenous data (e.g., from satellite or from NEXRAD) can be stored and merged with NetRad-generated data.

D. *Utility, Prediction, and Task Generation*

Within the feature repository, each voxel and each object has an associated utility that represents the "value" of scanning that voxel/object during in the next heartbeat. The utility value weights considerations such as the time since the voxel/object was last scanned, the object type (e.g., scanning an area with a tornado vortex will have

higher utility than sensing clear air), and user-based considerations such as the distance from a population center (e.g., among two objects with identical features, the one closer to a population center will have higher utility). We are currently developing a predicting component (shown with dashed lines in Figure 1) that tracks meteorological phenomena (e.g., a storm's centroid) and predicts their future locations. New objects, corresponding to the predicted future locations of the phenomena, can then be added into the feature repository – allowing these predicting modules to be easily integrated into the current architecture. In section 3, we present measured execution times of several different algorithms for storm-cell centroid tracking. The MC&C task-generation component takes objects from the feature repository and produces tasks, with an associated utility, for the optimization component. We use K-means clustering [Jain 1999] to generate the tasks. The initial centroids of the clusters are chosen by sorting the objects by utility and using the K spatially-separated objects with the highest utility as our starting points. This simple pre-clustering step is designed to ensure good spatial coverage for our clusters. The K-means distance metric uses a 4-dimensional vector of parameters: an objects X, Y position, utility, and meteorological type. The relative weighting of these parameters can be adjusted to give differing emphasis to each parameter. After clustering is complete, a final filtering step removes tasks with utility below a given threshold.

E. Optimization

The input to the radar targeting components is a list of objects that can potentially be scanned and their associated utility. The optimization module determines, for each radar, the angular sector to be scanned (targeted) by that radar for the next 30-second cycle. The overall utility of a given configuration of the radars, will depend not only on the utility of the objects scanned, but also the size of the sector (since larger sectors imply less time spent sensing a given radial) and the number of radars targeting a given volume (since more radars illuminating a volume implies higher accuracy of measurement). For our first testbed, a simple "brute-force" approach towards optimization is used that enumerates all possible configurations and computes the associated overall utility. In section 3, we present measured execution times for this approach. Other, more scalable, approaches towards optimization are currently being investigated.

III. BENCHMARKING OF INDIVIDUAL MC&C COMPONENTS

In this section, we present empirical measurements of the execution times of various NetRad MC&C components highlighted in Figure 1. All measurements were performed on a PC (3.2 GHz Intel CPU, 1 GB RAM) running Linux (RedHat 2). NetRad MC&C component execution times will depend on the radar data ingested by the system. Since, the NetRad system is not completely

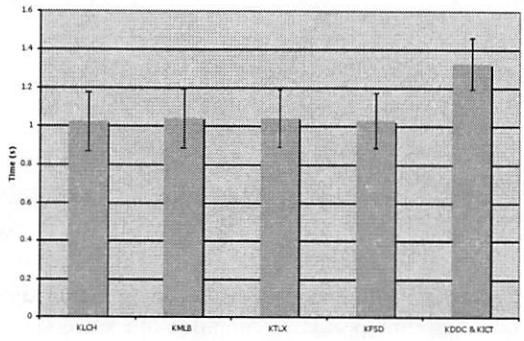


Figure 3: LDM to LDM, NetCDF file creation, event posting

build or deployed, we use existing NEXRAD [NOAA 2005] radar data and other sources, as described below, as input to the NetRad MC&C components. Recall that NEXRAD radars operate in "sit and spin" mode and thus the data ingested in one time period has no influence on the radars' scan strategy in a subsequent time period. Before presenting component runtimes, let us describe the radar data inputs used. Unless otherwise noted, we use the following six NEXRAD radar data sets, which provide per-radial range-gate reflectivity and wind velocity data. The data sets can be obtained from [CASA NEXRAD-data 2005]. The data in cases 1-4 are generated by a single radar; in cases 5 and 6 data comes from two partially overlapping radars. Each data set consists of sets of elevation scans, with a set of scans taken every five minutes. These sets of elevation scans are the input to our MC&C algorithms.

Test Case	Location	Date of event	Description of Events	Single/Multi Radar
1	KLCH Lake Charles LA	11/02 2004	Late season thunderstorm activity with scattered weak mesocyclones, tornadoes, waterspouts.	Single
2	KTLX Tulsa OK	5/03 1999	Extreme supercell outbreak including an F-5 tornado, costliest tornado in history.	Single
3	KFSD Sioux Falls SD	5/30-31 1998	Tornado outbreak with a number of vortices in close proximity to each other.	Single
4	KMLB Melbourne FL	9/05 2004	Hurricane Frances as it approaches Atlantic Coast of Florida at Category 2/3 with large, well defined eye and intense banding.	Single
5	KDDC Dodge City KS	6/06 2004	A strong bow-echo sweeping across the state of Kansas.	Multi
6	KICT Wichita KS	6/06 2004	A view of the aforementioned bow-echo case from a radar located further from the event.	Multi

Table 1: NEXRAD input data sets

A. Data Transmission, Storage, Event Posting

We begin our benchmarking of the MC&C by considering the time needed to transfer radar data from a

remote radar to the SOCC, store the per-elevation data in NetCDF format at the SOCC and post a notification event on the LB. Our measurements show that approximately one second is needed to compress and transfer a per-elevation scan from radar to SOCC over a Gigabit switch. We note that the nominal media transmission time (transmitting 20 KB of data into a Gigabit link) would increase from .00001 sec to .01sec if the link speed were changed to 1 Mbps. Thus, computing times rather than network media transmission times are the dominant factor here. We also note that these run times are for TCP transfers. We are currently developing a radar transport protocol that uses recent throughput measurements to avoid a slow-start phase [Schmitt 2002], and an application-specific congestion control/packet drop protocol [Banka 2005], as discussed in section 3. Once data arrives at the SOCC, reflectivity and velocity data files are created and an event is posted. Figure 3 shows that less than a second of runtime is needed to perform these operations.

B. *Detection Algorithms*

Figure 4 shows the per-elevation scan runtime needed by the LLSD algorithm. The standard deviation of the runtime is shown by the lower-valued super-imposed bar. LLSD requires less than 0.3 seconds per elevation scan on average, with the runtime decreasing with an increasing elevation angle. This behavior results from the fact that radar beams e aimed higher in the atmosphere, see decreased meteorological activity. Figure 5 shows the *merge* runtime for the 6 test cases. We took measurements for all six single radar cases and the additional case where

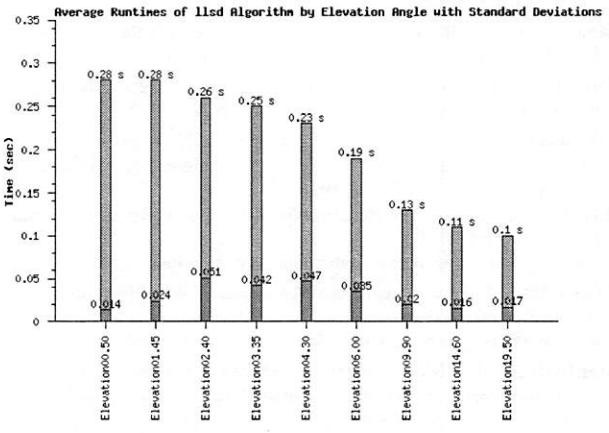


Figure 4: LLSD Run Time Per-Elevation Scan

the data for KICT and KDDC is merged. These radars have an overlap of roughly 50% percent. The results show that the runtime increases by approximately 30% with this amount of overlap. The difference between the single radar and the multiple radar case (KICT/KDDC) is that both coordinate conversion and data fusion must be performed for the overlapping region. Again, we see sub-second runtimes for this component.

C. *Task Generation, Prediction*

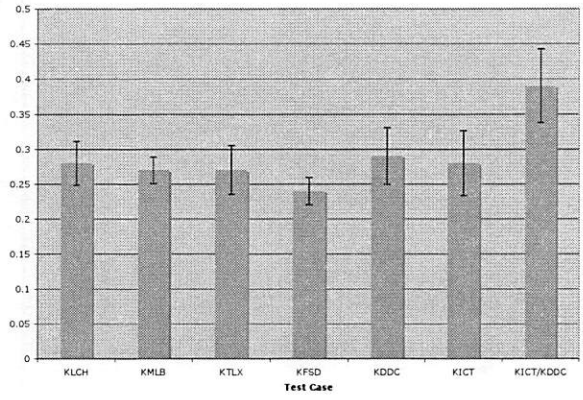


Figure 5: *merge* runtime, single and multi-radar cases

Recall from our earlier discussion, that the input to the task and utility generation routines is a multi-level grid of lat/long reflectivity and wind velocity values, with higher level "objects" superimposed on this grid. In order to use NEXRAD data in a NetRad MC&C environment, we assume that the NEXRAD data covers an 80km x 80km area. Two different timing values are collected. The first is the run time of the K-Means algorithm, the second is the time taken posting features to the feature repository. Additional data collected includes the number of points being clustered and the number of iterations required before the K-Means stabilizes on a set of clusters. Figure 6 also shows the runtime for hypothetical scenarios for coarser and finer grid sizes. We have also investigated the computational requirements of three approaches towards storm cell tracking: the WDSS II SCIT algorithm [Johnson 1998], a simple Kalman filtering algorithm, and a switched Kalman filtering algorithm A comparison of the tracking performance of these three algorithms is beyond the scope of this paper; see [Manfredi 2005] for details. Here, we note each algorithm requires less than 30 msec of execution time to perform one-step prediction, over 35 storm cell centroid tracks provided by NSSL.

D. *Radar Scanning Optimization*

The final step in closing the control loop is for the optimization module to determine the sectors to be scanned by the radars. The execution time of the optimization algorithm will depend on the number of radars, the extent to which the radars overlap, and the number/location/size of the meteorological objects in the radars' field. Figure 7 plots the average run time and standard deviation for the optimization module under several different scenarios, using the KFSD data. Along

the x-axis we vary the number of radars symmetrically placed in the 80x80 grid. We control the amount of overlap of the radars' footprint by changing the radius of each radar's circular footprint. Three runtime curves are plotted for the case of zero overlap (the edges of the

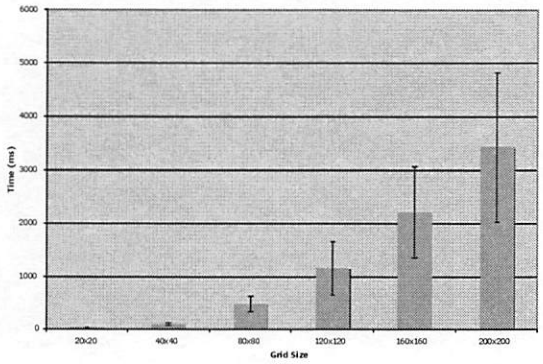


Figure 6: Task generation processing time

radars' footprint are coincident but non-overlapping), 33% overlap, and 67% overlap. For the case of 4 radars placed on 80km x 80km grid with 33% overlap, we see that the expected runtime is approximately 30 ms.

IV. Conclusions and Outlook

In this paper, we have described the software architecture of the meteorological command and control (MC&C) component of a NetRad prototype system currently under development – a dense network of low-powered radars that collaboratively and adaptively sense the lowest few kilometers of the earth's atmosphere The MC&C performs the system's main control loop – ingesting data from remote radars, identifying meteorological features in this data, reporting features to end-users, and determining each radar's future scan strategy based on detected features and end-user requirements. Our initial benchmarks, obtained by evaluating NetRad components using NEXRAD data show that that these components generally have sub-second execution times, making them well-suited for use in the NetRad system. Our future research will include the deployment and demonstration of the NetRad system, and continued enhancement of the detection, tracking, and optimization components. Thus far we have focused on a centralized view of the command and control architecture, which is appropriate given the relatively small number of radars in the initial testbed. However, several factors make control more difficult as the network scales, and preclude a purely centralized architecture. We are thus currently investigating distributed MC&C architectures that take advantage of the limited coupling among radars over large geographic areas.

V. Acknowledgements

We are grateful to Mark Simms for supplying scripts for processing WDSS II data, and Jerry Brotzge for many insightful conversations regarding the MC&C. This work was supported in part by the National Science Foundation under grant EEC-0313747 001.

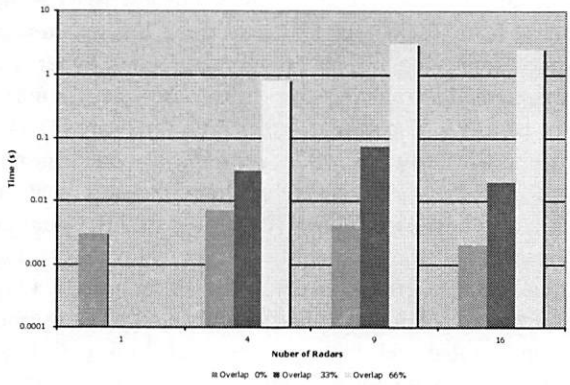


Figure 7: Optimization module runtime

VI. References

[Banka 2005] T. Banka, B. Donovan, V. Chandrasekar, A. Jayasumana, J. Kurose, "Data Transport Challenges in Emerging High-Bandwidth Real-Time Collaborative Adaptive Sensing Systems (poster)" to appear, *IEEE Infocom 2005.*

[CASA NEXRAD-data 2005] http://www.casa.umass.edu/eesr_data_sets

[Hondl 2002] K. Hondl, "Capabilities and Components of the Warning Decision and Support System – Integrated Information (WDSS-II), *Proc. American Meteorological Society Annual Meeting, Jan. 2003 (Long Beach).*

[Horling 2005] B. Horling, V. Lesser, "Distribution Strategies for Collaborative and Adaptive Sensor Networks.." *Proc. Int. Conf. Integration of Knowledge Intensive Multi-Agent Systems (KIMAS 2005).* April 2005. To appear.

[Jaganathan 1989] V. Jagannathan, R. Dodhiawala, L. Baum, *Blackboard Architectures, Applications*. Academic Press, 1989.

[Jaim 1999] A. K. Jain, M. N. Murty, P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264-323, 1999

[Johnson 1998] J. Johnson, *et al.*, "The Storm Cell Identification and Tracking Algorithm: an Enhanced WSR-88D algorithm," Weather and Forecasting, 13: 263-276, 1998.

[LDM 2005] University Corporation for Atmospheric Research, "Local Data Manager (LDM)." http://my.unidata.ucar.edu/content/software/ldm/index.html

[NOAA 2005] National Oceanic and Atmospheric Administration, "Radar Resources," http://www.ncdc.noaa.gov/oa/radar/radarresources.html

[Manfredi 2005] V. Mandfredi, S. Mahadevan, J. Kurose, "Kalman Filters for Prediction and Tracking in an Adaptive Sensor Network," Technical Report 05-07, CS Dept., U. Massachusetts.

[McLaughlin 2005] D. McLaughlin, V. Chandrasekar, K. Droegemeier, S. Frasier, J. Kurose, F. Junyent, B. Philips, S. Cruz-Pol, J. Colom, "Distributed Collaborative Adaptive Sensing for Improved Detection, Understanding, and Predicting of Atmospheric Hazards, *Proc. Ann. Am. Meteorological Soc. Mtg.*

[OneNet 2005] OneNet, "Oklahoma's Technology Network", http://www.onenet.net

[Schmidtt 2002] J. Schmitt, M. Zink, S. Theiss, R. Steinmetz, "Improving the Startup Behavior of TCP-Friendly Media Transmission," *Proc. Int. Network Conference (INC02).*

Reducing Business Surprises through Proactive, Real-Time Sensing and Alert Management

Mitchell A. Cohen, Jakka Sairamesh, Mao Chen
IBM T. J. Watson Research Center
Hawthorne, NY, 10532
{macohen, jramesh, maochen}@us.ibm.com

Abstract

OEMs need to transform the way they do business in order to ensure better quality of products and services. Crucial failure symptom information is lost between the end consumers of products and the manufacturers. Manufacturers have access to this information but are typically unable to handle its volume in a timely fashion. However, using this data properly can result in diminished labor time in issue resolution, decreased warranty costs for manufacturers and improved customer retention. In this paper, we present a novel system for Proactive Real-Time Event and Alert Processing, which enhances an enterprise's ability to monitor, analyze and detect critical business events and situations. This capability can help improve operational efficiencies, reduce costs, streamline processing of business alerts, and enable the enterprise to react in a more timely fashion. The system can enable monitoring of near real-time, low-level, industrial sensor and controller events, and high-level events from underlying structured and semi-structured data. The alert system uses domain knowledge to enable processing the events in real-time, performing the appropriate analytics and evaluation on the events and alerting the right set of users. The alert and event processing system has been deployed and validated in real pilots with industry specific data. We are currently validating the scalability and performance of the alert system with many different information sources and high volume sensor information.

1. Introduction

Over the last ten years we have been witnessing a transformation in the structure and daily operations of large and small enterprises because of the drive to lower costs, leverage the Internet for business operations, and integrate better with their supply chains. Increasingly, enterprises are dependent on their supply and value chains to build products and deliver services respectively. This has a profound impact as the streamlined flow of information and services in the supply and demand chains is crucial to manufacturers and enterprises wishing to capture the product feedback needed to improve product quality and increase customer satisfaction. For example, in the automotive industry, information disconnection in the demand chain is causing slower product feedback, rising warranty costs and declining customer satisfaction in spite of the advances in vehicle technology. Sensing and alert management in such demand chains can provide accurate and faster feedback, improve quality and reduce costs. In the following, we present the motivation for scalable sensing and alert management.

Rising warranty costs: It is estimated that in the US alone, the warranty costs in the automotive industry are around 3% of the OEMs (manufacturers) revenue, and the total costs for 2003 are estimated at 12 Billion USD[i], and rising. The same issues are being grappled with in the Electronics, Aerospace and Heavy Engineering industries. Manufacturers are extremely concerned about the rising warranty costs, and are determined to reduce them as much as possible through sensing product information (e.g., vehicles) and their status in real time.

Lack of visibility into operations: Most enterprises do not have deep visibility (at a fine grained level), controllable operational models and flexible and easily modifiable business operations. With sensors and actuators the right capabilities for monitoring operational performance, integration and control can be enabled for businesses to sense and adapt their daily operations in a timely manner. In the world of industrial sensors, the model of top-down design becomes critical as the right sensing of the operations can provide crucial information for optimizing business operations. The business metrics for enterprise operations can include improving the current cycle times of the core business processes in manufacturing, production, parts, inventory and sales. Shrinking the cycle times necessitates deeper and better visibility in

the enterprise operations. In this paper, we discuss various business scenarios, challenges and technology components that can enable such monitoring and evaluation in a scalable fashion.

Several papers and articles in the literature ([1], [2], [3], [4], and [5]) present novel event stream processing work. However, they have primarily focused on ad hoc communication domains or domain agnostic systems, and very few have focused on using semantics for event processing. The fundamental difference in our work is combining domain knowledge structures with sensor information to extract meaningful information in order to generate real alerts and take actions through complex decision processes or activate additional monitoring processes.

1.1. Business Metrics and Challenges

In order to enable monitoring of business operations, specific business metrics need to be defined for capturing the state of the business operations, current performance and trends. Tracking these metrics enables gathering of intelligence and predicting future trends in order to make timely decisions. For example, metrics are defined for managing the life-cycle of products from the production phase to the delivery and service phases. The following are typical examples of business metrics: a) Production and product life-cycle process times; b) Down time of assets in production and assembly lines; and c) Cost of monitoring production processes through automation. The challenging problems include identifying the metrics (assets or processes), sensing the data for the metrics in a scalable fashion, controlling the sampling rate dynamically, and evaluating the data to extract critical alert information.

1.2. Contributions

Summary of the Alert-Event System: In this paper, we describe a unified semantic event stream system that continuously monitors diverse data sources and generates alerts based on domain specific rules. This system can enable manufacturers to closely monitor critical business events (reducing surprises) and gather business intelligence from information such as product failures, warranty intelligence, field events, sales transactions, asset performance and others. Our solution is currently undergoing field trials and pilot deployments. This paper discusses the system design, methodology, reliability and scalability. A mathematical foundation is being worked on, but is beyond the scope of this paper.

2. Business Scenarios

In this section, we present two business processes and then describe the business and technical challenges.

2.1. Production, Engineering and Service

Production processes and asset-management: Enterprises are investing in sensor systems to enable the monitoring of the production processes, production assets and product lifecycles in the factory environment. The deployed sensors in the production environment capture daily production data and quality information in near real-time for the production engineers and specialists to keep a close watch on the manufacturing processes and assets involved in the process (e.g., robots on an assembly line).

Industrial Sector Demand Chain: In the Automotive Sector, manufacturers must manage rich relationships with dealers, fleets, and independent repair shops to support vehicle service lifecycles and build aftermarket revenue streams. Optimizing these aftermarket relationships requires OEMs to effectively support a wide range of critical business activities including real-time sensing of inventory replenishment, logistics, parts tracking, product behavior, retail performance management and others.

3. Technical Challenges

Real-time integration of diverse sensors and data types: Integration of a multitude of events and data from various sensors in various formats is a complex challenge. Sensors of various kinds are being deployed at every operational manufacturing site. These sensors are built by third-party vendors who have custom ways of sensing, sampling and generating the events. The data rates could range from 1-50 megabytes per second. The monitoring, integration, analysis and distribution of event information based on the data are critical for enterprises to streamline their manufacturing processes.

Programmability of controllers and actuators: Current pervasive systems are focused towards light-weight middleware layers over mobile devices with programming frameworks such as *MIDPs, SMF, OSGi, J2ME* and others. For a service oriented model, where sensor controllers and actuators offer services to higher level applications access (e.g., sampling rate), new models of programmability with messaging controller interfaces (e.g., Web Services) are needed.

Service-Oriented abstraction: With heterogeneity of various applications, information sources, sensors, and

sensor controllers, there is a need to abstract away the underlying sensor components and interfaces into a collection of services for higher level applications to use and configure. Each device controller can be defined by a service interface to enable regional or local computing servers to access and integrate with the controller.

4. Event Stream Processor Overview

At the center of the system solution is an Event Stream Processor which handles all events, ultimately deciding on the actions that need to be taken. Every kind of message coming from external systems, sensors and devices are treated as events. When running, the Event Stream Processor steps through following for each event: a) Event message receipt; b) Event transformation; c) Metric calculation; d) Metric evaluation; and e) Action invocation. The first 2 steps are performed at a message adapter layer.

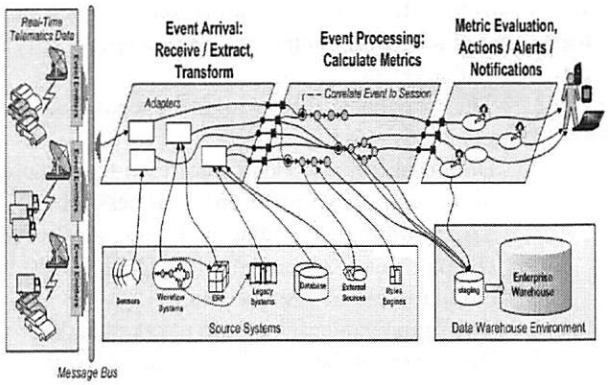


Figure 1. Flow of Event Stream Processing

4.1. Event Message Receipt

The Event Stream Processor has adapters for receiving messages. Adapters determine the transport mechanism for the group of senders (HTTP, JMS, etc). We have created adapters which have web services that can be called from clients. We also have adapters which can pull events by polling a potential event source. Here web services can be used to.

4.2. Event Message Transformation

Each adapter is capable of handling different formats of event messages. During transformation the messages are converted into a common format. Many different transformations are supported. The preferred method of transformation is via a web service call. The actual transformation can be done via any of the commonly accepted transformation techniques including XSLT, translator packages, and straight Java code.

4.3. Metric Calculation

Metrics are calculated measures based on incoming events. Metric calculations can go beyond the scope of events, potentially accessing other systems for additional data input or for additional calculations. Rules engine integration falls into the calculation category. Computations can range from mathematical calculations to text analytics including syntax and semantic analysis. For the automotive industry, we have created metrics for: a) Fleet management or individual vehicle monitoring, e.g., "running average of vehicle speed"; b) Complaint management, e.g., "warranty claim symptom"; and c) Class of vehicle performance, e.g., "percentage of vehicles within a class (Make, Model, Year, Trim combination) with warranty claims on a particular subcomponent."[ii]

Metrics can be processed on both structured and unstructured information. Telematics systems in automobiles collect many (anywhere from thirty to over 100) different parameters from vehicles. The telematics systems take snapshots of how the vehicle is being used and is performing. Values such as engine speed, vehicle speed, and tire pressure are collected multiple times per second. These values are well-structured real numbers. Metrics based on telematics data can be as simple as the value itself, such as "tire pressure" or can be calculations based on current and previous values, such as "running average of the vehicle speed to engine speed ratio." Calculations can include multiple different types of values, common with ratios, or even values from external sources.

Unstructured information may come in on events which contain text fields. For instance, automobile warranty claims come from dealers containing service technician write-ups. The text fields contain brief descriptions of the symptom and cause of a problem as well as the action taken. Each of symptom, cause and action can be metrics that are "calculated." In this case the calculation consists of syntactic and semantic analysis of the text using glossaries of categorized terms, abbreviations, and common misspellings.

4.4. Metric Evaluation

Prior to the Event Stream Processor being run, an expert (or a group of experts) with a great knowledge of the data, expected event arrivals, and normal overall system behavior creates rules based on the metrics. These rules are used in both the Metric Calculation and

Metric Evaluation phases of the processing. An example of such a rule on (structured) telemetric data in written form is "the twenty minute running average of vehicle speed should not exceed 75 miles per hour." Rules are actually stored in XML form. An example of a rule on the (unstructured) text of warranty claims using metrics described previously could be "there should never be a warranty claim where the symptom was fire" which will enable a quality analyst to be immediately alerted if such an unfortunate incident should occur.

Figure 2 shows each of the steps taken by the Event Stream Processor. Each step has a well-defined interface that is implemented by a local web service. Using such a Service Oriented Architecture allows easy modification of the processor's behavior through simple plug-and-play.

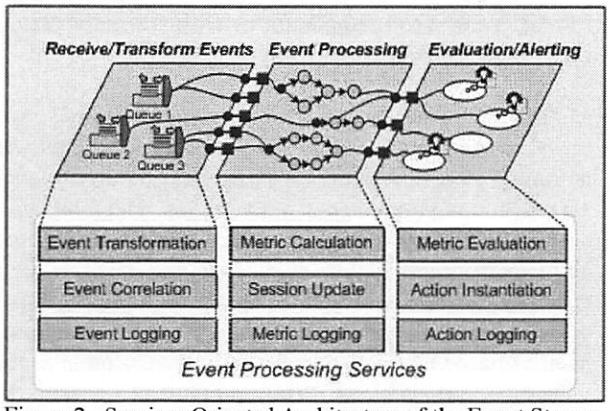


Figure 2. Services Oriented Architecture of the Event Stream Processor

Upon arrival, events are passed to the *Event Transformation Service* to be converted into the standard format used by the rest of the event processor. Based on the incoming event, the *Event Correlation Service* retrieves a list of metrics which need to get calculated. For a metric involving past values, the service also retrieves the relevant session. The values contained in the event are then used in calls to the *Metric Calculation Service*. The Event Stream Processor invokes the *Session Update Service* to update the session to ensure the current event is included in future metric calculations. The *Metric Evaluation Service* determines what actions if any need to be taken based on the newly calculated metric. These actions are taken with calls to the *Actions Instantiation Service*.

5. Scalability

Scalability is a key design point for the Event Stream Processor. The ability to handle large numbers of events has become paramount in the new world of sensors and actuators. For scalability, we need to look at the Event Stream Processor as well as the systems it uses. For the processor itself, there are two issues preventing simple replication of the server to allow any number of instances of it working in parallel, i.e., direct scalability. The preventative issues are session management and event listening.

The Event Stream Processor continuously calculates metrics, which get updated as events arrive. Calculations of metrics often depend on having sessions. That is, we need to store information about past events to calculate metrics based on new incoming events. A common example is with the calculation of running averages. Consider the metric of a running average of engine speed, a common value supplied by telemetry, for a particular vehicle. If the running average is over a ten minute period, then all data coming in over the last 10 minutes must be stored. All the data for a particular vehicle is needed together in the session to be able to do the calculation. As each new triplet of vehicle, timestamp, and engine speed arrives for a particular vehicle:

1. The session for the engine speeds for that vehicle is retrieved.
2. Data no longer needed (older than ten minutes in this case) is removed from the session.
3. The incoming timestamp and value are stored. (Steps 2 and 3 can be implemented with a circular array of session data.)
4. The running average metric is calculated.
5. Rules are applied to the calculated metric to determine if a warning needs to be indicated.

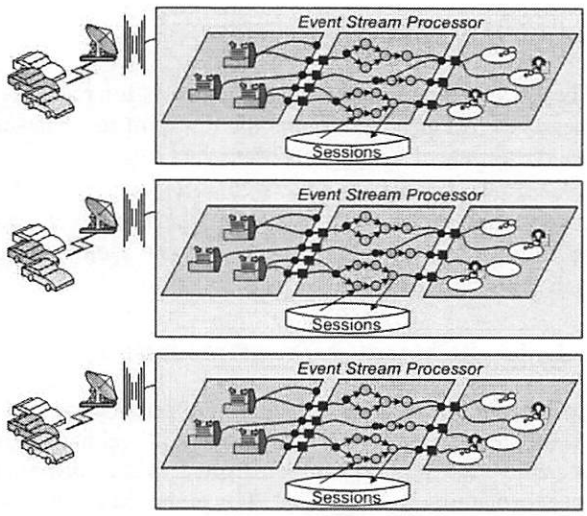


Figure 3. Assigning Event Creators with Their Own Processors with Local Sessions

Scalability can only be enabled by ensuring the current version of the session is available at the instance processing the event. One option for solving this problem is to have a common store for the sessions. As shown in Figure 4, the sessions can be stored in a Database Management System or a data grid. Each instance of the Event Stream Processor would retrieve and update the sessions as needed. Both database managers and data grids are often wrapped in web services. Another option for ensuring session availability, shown in Figure 3, is to assign each session to a particular Event Stream Processor instance. Sessions can then be stored and managed locally by each processor.

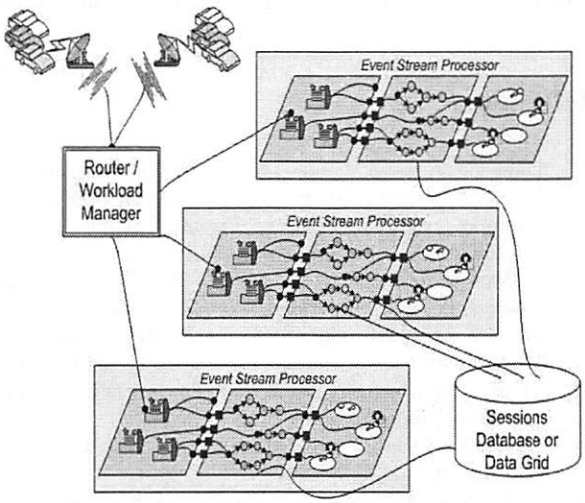


Figure 4. Using a Router to Pass Events to Processors with Shared Sessions

Message receipt remains separate from event processing allowing multiple nodes to handle the processing. In one instantiation of our system, we route incoming events based on the session to which they belong. Each event processing node handles the events for a group of sessions. There are two main factors in deciding how to partition the sessions across the event processing nodes:

1. Can we separate the sessions in a way so that all the metric calculation can be done locally? That is, can the data existing within the Event Stream Processor be collocated within nodes so that metric calculation can be done locally?
2. Can we separate the sessions so they are split up so that messages of the same type go to different event processing nodes? Messages are typically skewed by type. For instance, if metric calculations are mostly at the vehicle level, "change in transmission gear" events are much more common than "engine on" messages. Routing messages by vehicle will avoid workload problems caused by the difference in the volume of messages by message type.[iii]

6. Routing

There are two methods for handling the message routing:

1. Using a conventional message router
2. Assigning target web services for each of the different sessions

Message routers are well understood, but we'll explain further on the use of web services. Each different session can have its own web service. Each event creator can then be assigned a specific web service to call based on the session needed for its events. This type of mapping of event to sessions only works when sessions are only needed at the creator granularity. The metrics being calculated dictate the granularity. As long as a group of sessions rely only a specific set of event creators, those creators can use the same web service.

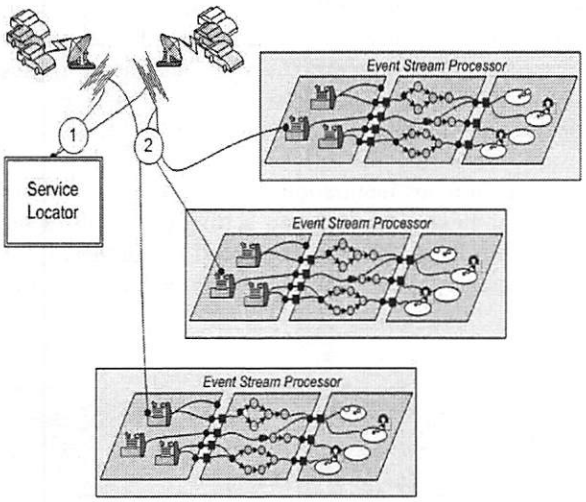


Figure 5. Using a Service Locator to Allow Event Creators to Send to a Processor

A more elaborate scheme to allow creators to send different types of events to different nodes consists of two steps. Prior to sending the event, a service is called to determine where to send the event. This routing decision can be based on the message itself. This mechanism pushes some of the routing processing onto the event creator. This approach has been described extensively in The Integrated Building Design Environment by Fenves et al [6]. A key ability of the Event Stream Processor is how at each step in its event

processing, it can integrate with external systems. In one particular instance we integrate with a semantic text analytics engine. Every time a warranty claim event is processed, it is passed along to the semantic text analytics engine to pull out symptoms, causes, and actions from text entered by the service technician at the automotive dealership. The integration can be done synchronously or asynchronously as shown in the figures below.

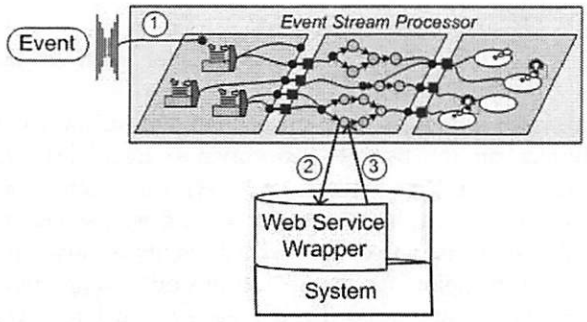


Figure 6. Synchronous Integration of External System

With synchronous integration, as shown in Figure 6, during event processing the call to the external system is made and the processor thread making the call awaits a response. Choosing synchronous integration makes most sense when integrating with external systems that respond quickly relative the needed response time.

The asynchronous integration shown in Figure 7 allows processing to continue prior to getting the result of the call. When the processing of the call is completed, the system creates a new event which allows processing to continue.

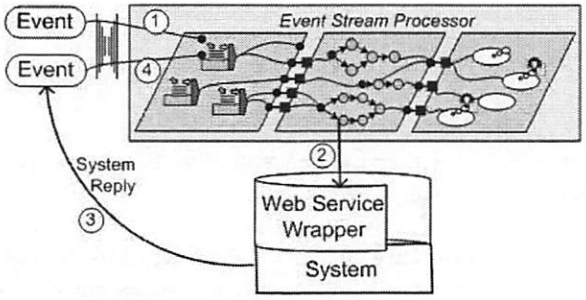


Figure 7. Asynchronous Integration with an External System

7. Conclusion

In this paper, we described a novel unified semantic event stream processing system for general critical alerts that enhances manufacturer process efficiency in quality, service, and warranty management. This system improves visibility into critical alerts and reduces surprises on warranty costs. It allows business early access to information and trends. Our alert and event system is undergoing field trials. A version of the solution has already been deployed and functional on real industrial data. The system is written in Java with XML and Web Services as the support for semantic data structures, rules, configuration and evaluation. The performance has been excellent on real-time information streams. Flexibility allows multiple deployment scenarios enabling the system to be optimized for various event evaluation mechanisms. Various techniques are deployed to allow for scalability with the metric types and data patterns determining which technique is to be used. Further validation on the scalability and routing is being done in real pilots and engagements.

References

[1] Daniel J Abadi et. al., *The Design of the Borealis Stream Processing Engine*, Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA, January 2005.

[2] C. McGregor and Josef Schiefer, *A Web-Service based framework for analyzing and measuring business performance*, Information Systems and E-Business Management, Volume 2, Issue 1, Apr 2004, pp. 89-110.

[3] Tivoli Risk Manager, Event Correlation Engine, 2003. http://www.ibm.com/tivoli.

[4] David Luckham and Mark Palmer, *Separating the Wheat from the Chaff*, RFID Journal, 2004.

[5] Rajit Manohar and K. Mani Chandy, *Dataflow Networks for Event Stream Processing*, 16th IASTED International Conference on Parallel and Distributed Computing and Systems, November 2004.

[6] S. Fenves, U. Flemming, C. Hendrickson, M. Maher, R. Quadrel, M. Terk, and R. Woodbury. *Concurrent Computer-Integrated Building Design.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 07632, 1998.

[i] Estimate is from AMR Research – on the web at http://www.advmfg.com/content/resourcecenter.asp?id=440.

[ii] Automotive *components* include high level systems such as Engine, Transmission, and Body. *Subcomponents* are subsystems within the components. For instance, a subcomponent within the Engine component is the High Pressure Oil System.

[iii] Of course, there can be skew in the messages created for different vehicles. A whole paper (or perhaps even a text book) can be written on dealing with message skew.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

For a complete listing of member benefits, see http://www.usenix.org/membership/bens.html.

Want more information about USENIX? See http://www.usenix.org/ or contact:

USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA
Phone: 510-528-8649 Fax: 510-548-5738 Email: *office@usenix.org*

Thanks to USENIX Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ Ajava Systems, Inc. ❖ AMD ❖ Asian Development Bank ❖
❖ Atos Origin BV ❖ Cambridge Computer Services, Inc. ❖ Delmar Learning ❖
❖ DoCoMo Communications Laboratories USA, Inc. ❖ Electronic Frontier Foundation ❖
❖ Hewlett-Packard ❖ IBM ❖ Intel ❖ Interhack MacConnection ❖ The Measurement Factory ❖
❖ Microsoft Research ❖ ❖ Oracle ❖ OSDL ❖ Perfect Order ❖ Portlock Software ❖ Raytheon ❖
❖ Sun Microsystems, Inc. ❖ Taos ❖ Tellme Networks ❖ UUNET Technologies, Inc. ❖ Veritas Software ❖

ACM SIGMOBILE

ACM SIGMOBILE is the Association for Computing Machinery's Special Interest Group on Mobility of Systems, Users, Data, and Computing. Founded in 1947, ACM is the world's first educational and scientific computing society. Today, ACM serves a membership of more than 80,000 computing professionals in more than 100 countries in all areas of industry, academia, and government.

SIGMOBILE is the primary international organization dedicated to addressing the latest developments in all areas of mobile computing and wireless and mobile networking. SIGMOBILE has members from around the world, from academic organizations, industry research and development, government, and other interested individuals. Members of SIGMOBILE are active in the development of new technologies and techniques for mobile and wireless computing and communications.

As a member of SIGMOBILE, you will receiver our quarterly journal, *Mobile Computing and Communications Review* (MC R). You will also be able to receive announcements via our moderated members-only email distribution list, keeping you informed of the latest happenings in our field, including new opportunities to share your research and to meet with friends and colleagues at conferences and other events. SIGMOBILE members also are eligible for the lowest generally available registration fee for any event sponsored or co-sponsored by SIGMOBILE, and for the many events sponsored by other organizations offered in-cooperation with SIGMOBILE.

For more information about ACM SIGMOBILE, visit us on the web at *http://www.sigmobile.org/*. You may also contact the ACM Membership Services Department by email at *acmhelp@acm.org* or by telephone at 1-800-342-6626 (U.S. and Canada) or +1-212-626-0500 (outside U.S.).

ISBN 1-931971-32-3